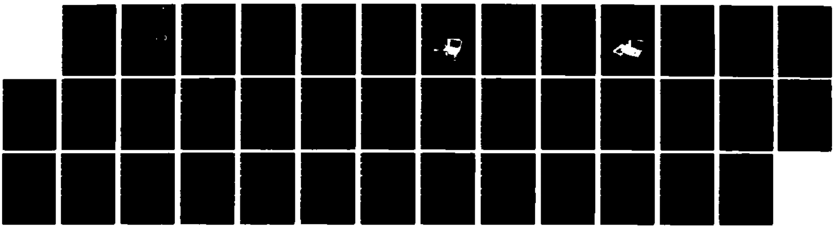
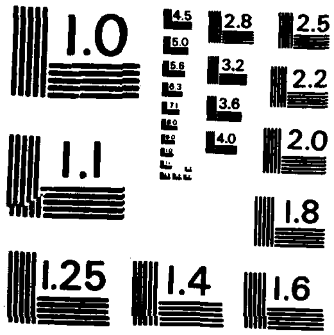
AD-A166 470 USACRREL (US ARMY COLD REGIONS RESEARCH & ENGINEERING LABORATORY) PRECISE. (U) COLD REGIONS RESEARCH AND ENGINERRING LAB HANOVER NH G M TRACHIER ET AL. DEC 85 1/1

UNCLASSIFIED CRREL-SR-85-26 F/G 14/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Special Report 85-26

December 1985

US Army Corps of Engineers
Cold Regions Research & Engineering Laboratory

(12)

USACRREL precise thermistor meter

G.M. Trachier, J.S. Morse and S.F. Daly

Approved for public release; distribution is unlimited.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Special Report 85-26	2. GOVT ACCESSION NO. ADA166470	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) USACRREL PRECISE THERMISTOR METER		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) G.M. Trachier, J.S. Morse and S.F. Daly		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS U.S. Army Cold Regions Research and Engineering Laboratory Hanover, New Hampshire 03755-1290		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS DA Project 4A161101A91D
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE December 1985
		13. NUMBER OF PAGES 39
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Frazil ice
Ice
Instrument fabrication
Water temperature measurement

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

To facilitate the study of frazil ice in the field, a highly accurate, portable water temperature meter was required. The USACRREL Precise Thermistor Meter was designed and built to meet this need. The meter is rugged, battery-operated, waterproof, and able to operate over a wide range of ambient temperatures. A unique feature of this instrument is the use of software to compensate for temperature-dependent variation in the analog electronics. The circuitry consists of an analog printed circuit board and a low power microcomputer. The resistance of a calibrated thermistor is determined and its temperature calculated using the Steinhart-Hart equation. The accuracy of the meter was determined

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract (cont'd).

both theoretically and in cold room tests. The hardware and software used in the meter are described.

PREFACE

This report was prepared by Gary M. Trachier, Electronics Technician, James S. Morse, Electronics Engineer, both of the Engineering and Measurement Services Branch, Technical Services Division, and Steven F. Daly, Research Hydraulic Engineer, Ice Engineering Research Branch, Experimental Engineering Division, U.S. Army Cold Regions Research and Engineering Laboratory. Funding was provided by DA Project 4A161101A91D, In-House Laboratory Independent Research.

Steven F. Daly determined the instrument's capabilities and requirements. James Morse conceived the idea of interfacing a thermistor and resistance-to-voltage converter to a microprocessor. Gary Trachier selected the modules and components, built the instrument, developed the software and tested the final product. The authors thank John Kalafut for his assistance in building the instrument, Robert Demars and David L'Heureux for their precision photography and assistance in making the printed circuit boards, Mark Hardenberg for editing this report and Matthew Pacillo for drafting the figures.

The contents of this report are not to be used for advertising or promotional purposes. Citation of brand names does not constitute an official endorsement or approval of the use of such commercial products.

CONTENTS

ILLUSTRATIONS

TABLES

USACRREL PRECISE THERMISTOR METER

G.M. Trachier, J.S. Morse and S.F. Daly

INTRODUCTION

The study of ice in rivers, and particularly frazil ice, has been frustrated by the lack of an instrument capable of determining water temperature with sufficient accuracy and precision. For example, in the study of frazil ice produced in rivers and streams, it is well known that supercooling of the water during frazil production in the field rarely exceeds 0.03°C, and never exceeds 0.05°C. It is vitally important that temperatures of supercooling be measured accurately to within 0.01°C for us to increase our understanding of the fundamentals of frazil formation.

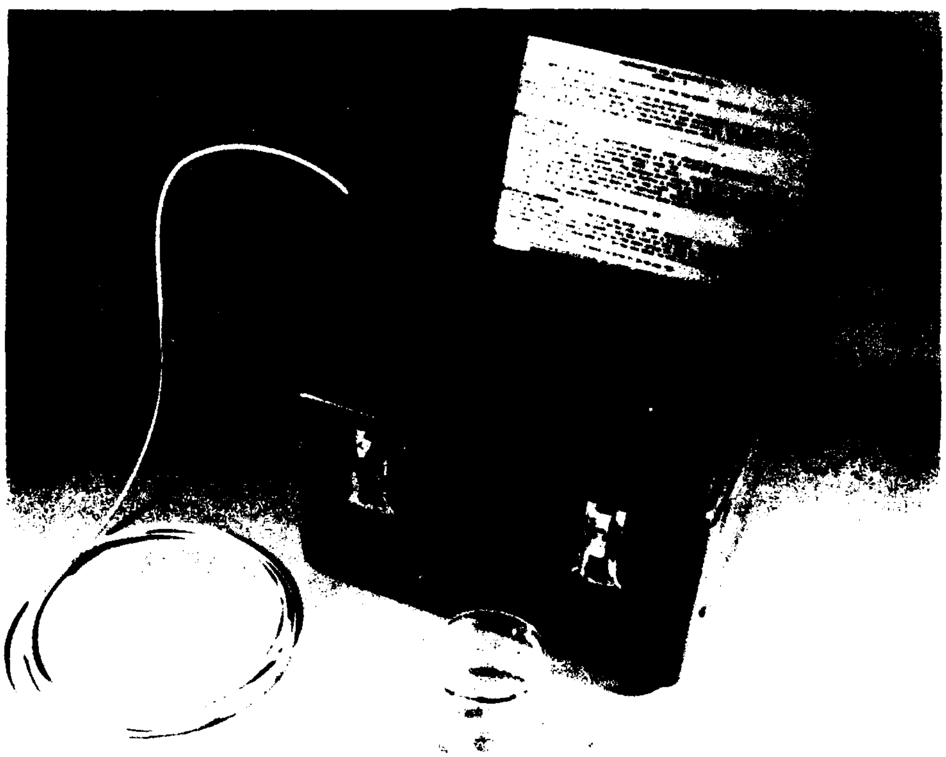


Figure 1. USACRREL precise thermistor meter.

Many other ice phenomena may also be critically dependent on small changes of supercooling. However, until now there has existed no temperature measuring device that is both portable and rugged enough to be used in the field and that has the precision and accuracy necessary.

This report describes an accurate, precise temperature measurement instrument recently developed at CRREL (Fig. 1). A unique feature of this instrument is the use of software to compensate for temperature-dependent variation in the instrument's circuitry. This greatly improves the accuracy of the instrument over a wide range of ambient temperatures.

INSTRUMENT REQUIREMENTS

The meter would be primarily used to measure water temperature under a variety of field conditions. It had to be built to meet the following requirements:

1. Accuracy -- the meter should be accurate to 0.01°C ± 0.025°C.
2. Operating conditions -- the meter should be able to operate under a variety of field conditions in temperatures ranging from -30° to 20°C. It must also be waterproof, buoyant, and rugged enough to withstand the rigors of field use.
3. Usability -- the meter should have a digital readout of temperature. It should be hand-held and convenient to use. To allow maximum portability, it should be battery operated.

INSTRUMENT DEVELOPMENT

General

Given the meter requirements outlined in the previous section, a survey was made of the commercially available digital thermometers; none that approached the required accuracy and resolution were suitable for field use. The majority required a 110-Vac power source and all would operate correctly under a very limited ambient temperature range. In addition, none were rugged enough or waterproof. Therefore, we decided to develop and construct a unit at CRREL.

The choice of temperature sensors was narrowed down to either a Platinum Resistance Temperature Device (PtRTD) or a calibrated thermistor. Both are widely used in industry and known for their long-term stability. We chose the thermistor. Thermistors have a much larger change in

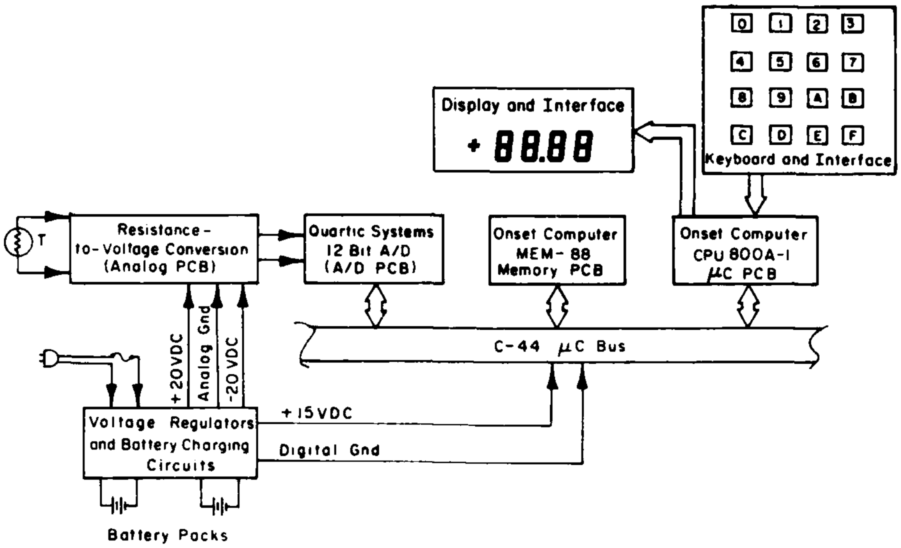


Figure 2. Instrument circuit block diagram.

resistance per unit temperature change than a PtRTD, allowing greater accuracy in measuring small changes in temperature.

There are two popular methods for precisely measuring resistance. One uses the Wheatstone bridge, where the unknown resistance is matched to a known resistance. The other uses a constant current that is passed through the thermistor while the voltage across it is measured. The resistance can then be calculated using Ohm's law. This is the principle behind a standard ohmmeter. These two methods are well proven, but inconvenient for determining temperature. The user must first determine the thermistor's resistance. He must then look up the resistance on a special table relating resistance to temperature. One of these tables must be on hand for each thermistor being used.

We decided to use a calibrated thermistor, constant current and an analog-to-digital (A/D) converter to measure the voltage (Fig. 2). This voltage would then be used to determine the thermistor's resistance. To avoid the inconvenience of conversion tables, we included a microcomputer in the instrument to convert the thermistor's resistance to a temperature reading. The calculated temperature is then displayed.

The known resistance of the thermistor can be used to determine its temperature by use of the Steinhart-Hart equation (Steinhart and Hart 1968, Yellow Springs Instruments Inc. 1971). The constants for the Steinhart-Hart equation would be entered by the user into memory, and remain there until changed.

The Steinhart-Hart equation, which describes the temperature of a thermistor as a function of its resistance, is

$$\frac{1}{T} = A + B * (\ln R) + C * (\ln R)^3 \tag{1}$$

where T = temperature (Kelvins)

R = thermistor resistance (ohms)

A,B,C = fitting constants (determined through the thermistor calibration).

Hardware

The electronics are housed in a case that is 7 in. wide, 11 in. long and 8-1/2 in. high, which can be hand carried. The case has been sealed thoroughly with silicone sealant on all through-holes, cutouts and weld seams. This includes sealing the joint between the top panel and the lower box, and around all mounting holes on the top panel. Note that the hinged cover does not make a waterproof seal when closed. The instrument's electronics consist of an analog Printed Circuit Board (PCB) that converts the thermistor's resistance to a voltage, and a low power microcomputer that converts the voltage to a temperature reading and displays it. The

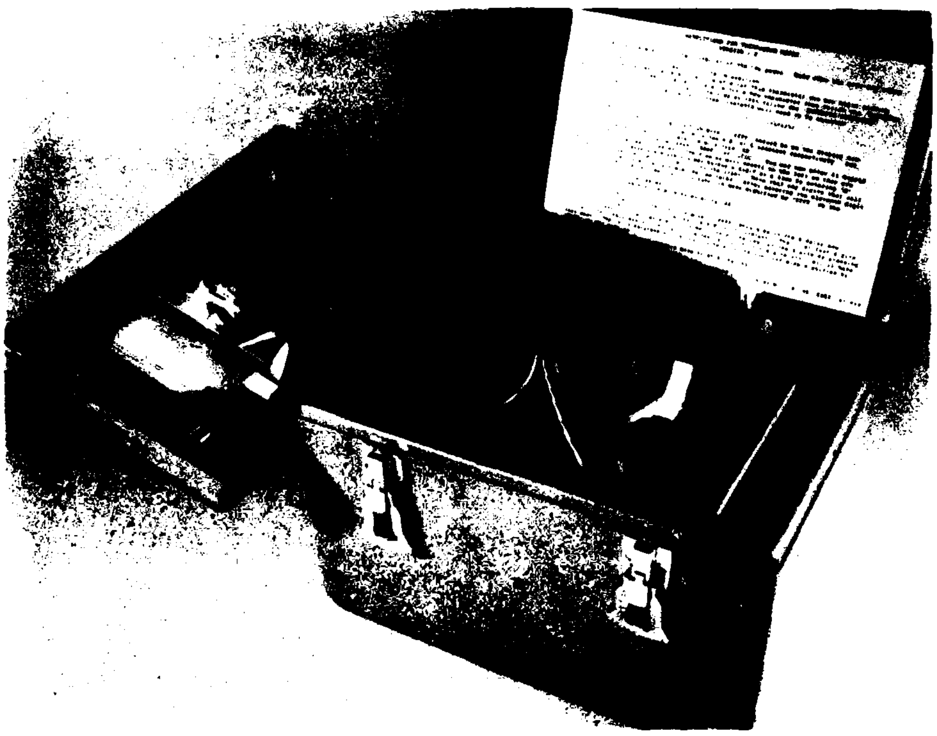
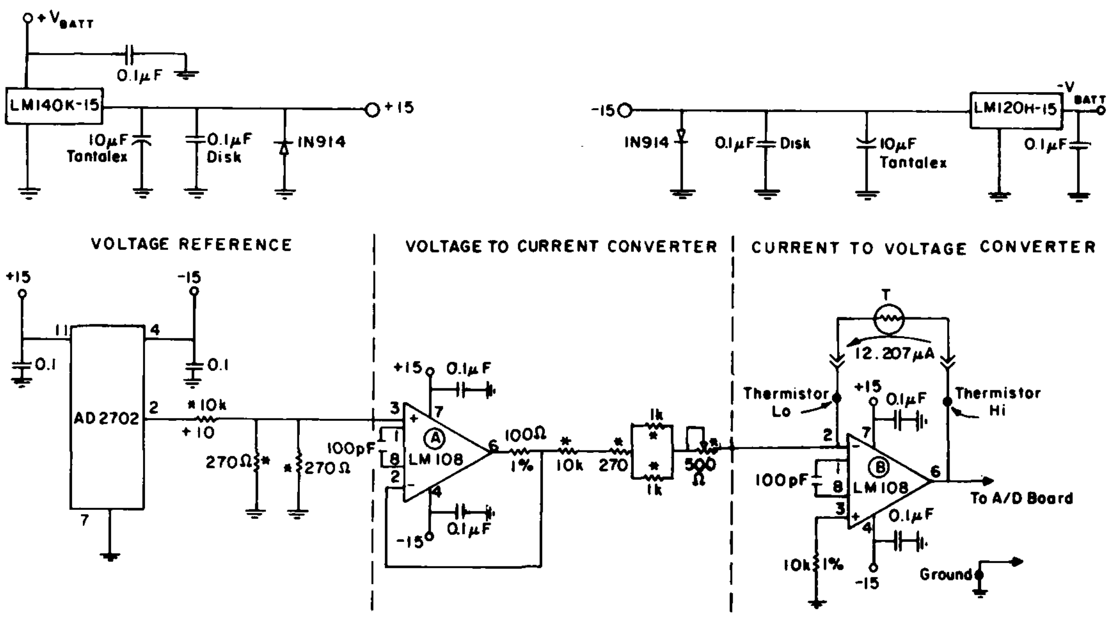


Figure 3. Hardware.



•Board edge test point.
*Low temperature coefficient part.

Figure 4. Schematic of analog PCB circuit (low temperature coefficient bypass capacitors used; a bypass capacitor is placed between the digital and analog grounds on the PCB).

microcomputer consists of a processor board, memory board and an A/D converter board. The microcomputer programs were created on a National Semiconductor Corporation Starplex II microprocessor development system. With this system, the microcomputer programs were written in a higher level language, compiled, interactively debugged and then "down loaded" to the microcomputer. Figure 3 shows the hardware.

Analog board

The schematic diagram of the analog PCB is shown in Figure 4. The analog PCB consists of the circuitry needed to convert the thermistor's resistance to a millivolt value and can be divided into two sections. The first portion of the circuit is a voltage-to-current converter. It is a standard "op-amp" (operational amplifier) circuit that converts the output of the precision voltage reference to a constant current source (Stout and Kaufman 1976). The AD2702UD precision voltage reference is the most stable one available without an internal heater. Its nominal output is 10.000 Vdc, with a low (5 ppm/°C) temperature coefficient. We wanted one without a heater to minimize power consumption from the battery pack. The LM108AHM operational amplifier also has a very low power requirement and temperature coefficient (1 μV/°C). All of the resistors in this part of the circuit have temperature coefficients of only 2 ppm/°C.

The last part of the circuit is a current-to-voltage converter. The thermistor is placed in the op-amp's feedback loop. This assures that a constant current of 12.207 μA is passed through it. The op-amp's output voltage is then linearly proportional to the thermistor's resistance (Stout and Kaufman 1976).

Microcomputer system

Processor board. The processor board controls all Input/Output (I/O), corrects the voltage for temperature-dependent variation, determines the thermistor's resistance, and solves the Steinhart-Hart equation to determine the thermistor's temperature. It has a National Semiconductor Corporation (NSC800) microprocessor, 2 kilobytes of Eraseable, Programmable, Read Only Memory (EPROM) and 2 kilobytes of Random Access Memory (RAM). It also contains the I/O ports that interface to the keyboard and Liquid Crystal Display (LCD). The machine cycle time of the microcomputer is 1 μs.

Memory board. The program and thermistor constants are stored on the memory board. Seven 27C16 2-kilobyte by 8-bit EPROMs and one 52B13 2-kilobyte by 8-bit Electrically Alterable, Read Only Memory (EAROM) are mounted on this board. The program is stored in the former and the thermistor fitting constants in the latter. The EAROM requires a 10 ms write cycle, so a hardware time delay was designed to lengthen the machine write cycle from 1 μs to 10 ms.

A/D board. The A/D board converts the millivolt output from the analog PCB to digital. A 12-bit CMOS integrating converter does this. There are eight differential input channels that can be selected individually and a programmable gain amplifier. The six ranges of the programmable gain amplifier have full scale input voltages of 5.0, 2.5, 1.0, 0.5, 0.25, and 0.1 V. Temperature-dependent variations of this circuitry are corrected by the software.

Only two of the eight available channels are used. One channel samples the output of the analog PCB and the other monitors an onboard temperature sensor, measuring the instrument's internal temperature. This

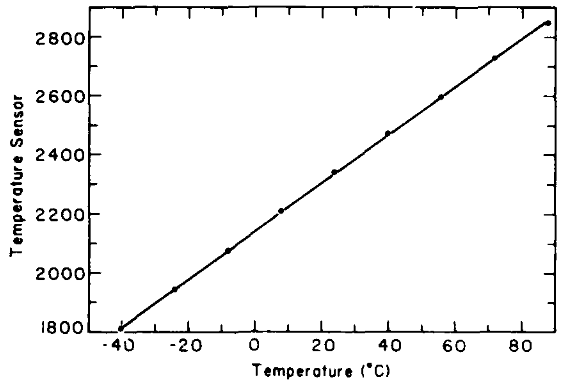


Figure 5. Output units of onboard temperature sensor.

measurement is required to allow the software to compensate for temperature-dependent variations in the analog PCB and the A/D board. The output from the internal sensor is proportional to the actual temperature. The relationship between the internal sensor output and temperature (°C) is shown in Figure 5.

Software

The software has two tasks. The first is to control the instrument itself, initializing the instrument when it is turned on, accessing the A/D converter, accepting information from the keyboard, displaying information, etc. The second is to accurately determine the temperature of the thermistor. It does this by applying the compensations required because of the instrument's temperature, calculating the thermistor's true resistance, and then solving the Steinhart-Hart equation. Appendix A contains listings and flowcharts for the software.

The software module that accomplishes the first task is labeled METER (Fig. A1a). METER was written in the higher level language PLM80. This language is very convenient for interfacing the several boards that form the instrument and it is especially useful for testing and manipulating the bits of data that must be transferred between the boards.

The software module that accomplishes the second task is labeled TMPCLC (Fig. A1b). This module is called as a subroutine by METER. TMPCLC was written in the higher level language PASCAL because it would have been impossible to do the required calculations with PLM80's 16-bit integer arithmetic. PASCAL, on the other hand, is very good for mathematical calculations but is difficult to use for interfacing boards.

As mentioned earlier, the instrument was developed on a National Semiconductor Corporation Starplex II microprocessor development system. However, all of the Starplex system calls, file structure subroutines and I/O subroutines were eliminated from the run-time library. This saves a great deal of memory space.

Combining the PASCAL and PLM80 languages presented special problems, including data transfer, duplicate function names and other interfacing problems. To facilitate data transfer, all information is passed between METER and TMPCLC through dedicated memory locations. This eliminates any need for matching variable types between the languages. Debugging is made easier when we know where each variable is stored.

METER

When the instrument is turned on, the program begins running METER. The first function is to initialize the I/O ports, the stack pointer and the PASCAL pointers. The A/D circuit board is also initialized. The program then repeatedly samples the A/D board. Sampling the A/D board for a valid reading is a two-step process. First, the programmable op-amp is set to the highest amplification, and second, a reading is taken on this range and checked for an over-range indication. If it is not over-range, the program returns to the main loop. If the reading was over-range the op-amp is set to the next lower amplification. The A/D board is sampled and checked again for an overrange condition. This process of reducing the amplification continues through each of the six ranges until a valid reading is obtained. This procedure is used to keep the number output from the A/D board as large as possible without going over-range, thus providing the maximum accuracy of the instrument. When a valid reading has been obtained, TMPCLC is called and the reading is converted to a temperature.

The keyboard is sampled within the range selection loop. If the 'F'-key is being pressed, the command interpreter subroutine labeled FUNCTION is called. All other keys will be ignored. There are two things the user may do while in FUNCTION. The thermistor calibration constants in use may be displayed or new ones entered (refer to Appendix B for more detailed operating instructions).

TMPCLC

The main function of TMPCLC is to convert the raw number output from the A/D converter to a temperature reading. First, TMPCLC reads the thermistor constants from the EAROM and combines the digits to form floating point numbers. It uses a lookup table to determine the op-amp's voltage range and the range-dependent temperature compensation coefficients.

The first compensation is applied to the full-scale voltage of the A/D board at the range selected. Through experimentation, described in the next section, we found that the full-scale voltage could be described as

$$VFS_n = a_n + b_n T + c_n T^2 + d_n T^3 \qquad (2)$$

where VFS_n is the full-scale voltage of the nth range, a_n, b_n, c_n, d_n are the temperature compensation coefficients associated with the nth range, and T is the instrument temperature, determined by the internal sensor.

Next, the raw binary output of the A/D board is corrected for temperature. This is done by the equation

$$Y = X_1 + m_n T + b_{on} \qquad (3)$$

where X_1 is the raw output of the A/D board, m_n and b_{on} are constants, associated with the nth range, and T is the instrument temperature, determined by the onboard sensor. This equation in effect applies a temperature-dependent offset shift to the raw output of the A/D board. This offset was found through experimentation, again described in the next section.

The actual voltage measured by the A/D board is found as

$$MVADC = (Y/4095) * VFS \qquad (4)$$

where MVADC is the actual measured voltage, and 4095 is the total number of bits at full scale. Next, there is a known offset associated with the analog PCB, which is temperature-dependent. This offset, F_s, is determined as

$$F_s = a_{PCB} + b_{PCB} T + c_{PCB} T^2 + b_{PCB} T^3 \qquad (5)$$

where a_{PCB}, B_{PCB}, c_{PCB}, d_{PCB} are the temperature compensation coefficients associated with the analog PCB, and T is the instrument temperature, determined by the onboard sensor. Finally, the true resistance of the thermistor, RES, can be calculated as

$$RES = MVTRUE/12.207E-06 \qquad (6)$$

where RES is the thermistor resistance in ohms, 12.207E-06 is the constant current passed through the thermistor, and MVTRUE = MVADC + F_s.

Next, the Steinhart-Hart equation (eq 1) is used to convert the thermistor's resistance to a temperature. At present, the program will work only with thermistors with about 5 kΩ resistance at 0°C, as the exponents of the A, B, and C coefficients are declared in TMPCLC as constants.

INSTRUMENT ACCURACY

The following sources of possible error have been identified.

Errors external to the instrument

Probe lead resistance

The probe leads will add resistance in series with the thermistor. This will cause a decrease in the apparent temperature of the sample. The 6-ft 18-AWG (American Wire Gauge) stranded probe wire provided with the instrument has a resistance of roughly 30 mΩ. This results in an apparent temperature decrease of about 0.001°C at an ambient temperature of 20°C, which is the worst case.

Self-heating of the thermistor probe

Self-heating is the increase in temperature of the thermistor from the dissipation of electrical energy within the thermistor itself. Calculations based on dissipation constants for bead thermistors (Omega 1984) show that for the worst case of still air, the temperature error is only about +0.003°C. For a thermistor in a well-stirred oil bath, the error is only +0.0004°C in the worst case.

Calibration accuracy of the thermistors

Measurement error during the thermistor calibration is another factor but is beyond the scope of this report.

Uncertainty of the Steinhart-Hart equation

It has been shown that if the temperature span between any two adjacent calibration points is less than 50°C, the Steinhart-Hart equation will reproduce the actual temperature within 0.01°C (Yellow Springs Instruments, Inc. 1971).

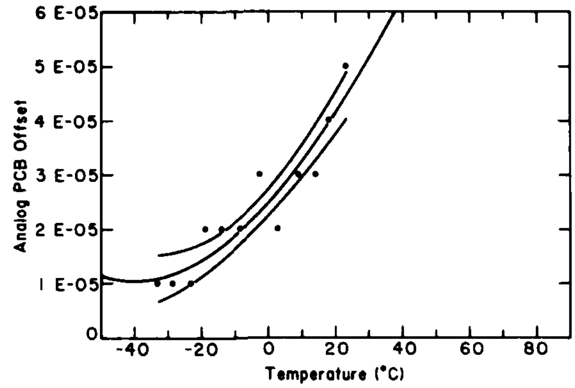


Figure 6. Analog PCB offset temperature compensation (center curve is eq 5, with 90% confidence band shown.)

Internal Instrument errors

The remaining sources of error are associated with the instrument itself. The individual errors have been characterized and, where possible, are corrected by the software. The temperature correction equations in TMPCLC play an important part in the instrument. Without them its accuracy would be far less. The following procedure was used to determine the temperature-dependent variation of the A/D converter board and the analog PCB.

Each was put into a cold chamber. The chamber's temperature was varied while the input to each board was held constant. The board output was measured and a regression done on the data relating the board's output to its temperature.

Temperature effects on analog PCB gain and offset

Through the tests described above we found that the offset (F_s) of the analog PCB was temperature dependent. Equation 5 determines the temperature-dependent offset of the resistance-to-voltage circuit. We found that offset error is much more significant than gain and nonlinearity errors. The latter two are small enough to be ignored (see Fig. 6 for the calibration curve).

Long-term stability of the analog PCB

The long-term stability of the circuit is both unknown and uncorrectable. The user should keep in mind, though, that most electronic instruments should be recalibrated at least once a year.

Accuracy of the PASCAL math package

National Semiconductor's PASCAL math package uses 24-bit floating point numbers internally. This maintains about 7 digits of precision. The resulting error is insignificant.

Round-off of displayed temperatures

The resolution of the liquid crystal display is 0.01°C. This means that the temperatures must be rounded to the nearest 0.01°C before being displayed. This contributes up to ± 0.005°C error. This error can be reduced only by using a display with more digits.

A/D nonlinearity

The analog-to-digital converter's nonlinearity is uncorrectable and contributes ± 1/2 bit to the instrument error.

A/D quantizing error

Quantizing error is present whenever there is converting between analog and digital. The uncertainty is always ± 1/2 of the least-significant-bit of the converter.

A/D input offset current

The analog-to-digital converter's offset is affected by the output impedence of the previous stage. A higher output impedance results in more offset. In this case the offset is negligible because the op-amp in the previous stage has a very low output impedence.

A/D nonideal gain and offset

Equation 2 compensates for the temperature-dependent gain change of the A/D converter. The nominal full-scale voltage ranges are 5.0, 2.5, 1.0, 0.5, 0.25, and 0.1 V; there is a set of coefficients for each of the six voltage ranges. The program uses a table to select the proper set of coefficients. Figure 7 shows the raw data and regression plot for each voltage range.

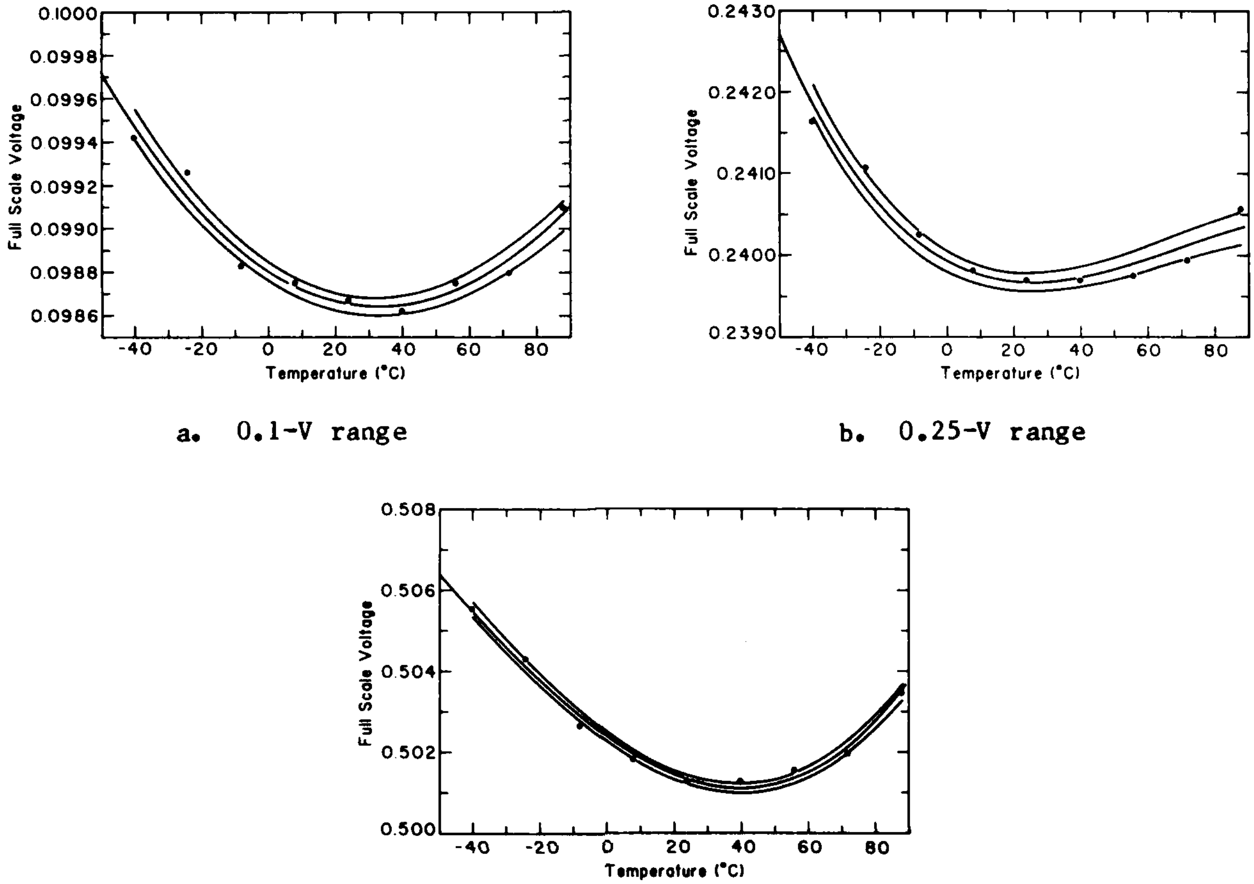
The raw output of the A/D conversion is corrected for offset shift by eq 3. This offset error is temperature dependent and can be approximated by a linear equation.

A/D long-term stability

The long-term stability of the circuit is both unknown and uncorrectable. The circuit board has not been in use long enough to be evaluated over the long term. The user should keep in mind though that most electronic instruments should be recalibrated at least once a year.

Instrument error analysis

An error analysis was performed for two assumed field conditions. The first was to keep the thermistor's temperature at 0°C while varying the



a. 0.1-V range

b. 0.25-V range

c. 0.5-V range.

Figure 7. Calibration curves for A/D converter (center curve is eq 2 with 90% confidence band shown.)

Table 1. Thermistor at 0°C with instrument temperature varied.

Error source	Temperature (°C) 20	15	10	5	0	-5	-10	-15	-20	-25	-30	-35
Display round-off	± 0.64	0.64	0.64	0.64	0.64	0.64	0.64	0.64	0.64	0.64	0.64	0.64
A/D nonlinearity	± 0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
A/D quantizing	± 0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
A/D non-ideal gain	± 1.60	1.64	1.67	1.74	1.76	1.93	2.04	2.18	2.31	2.45	2.61	2.76
Analog PCB offset	± 0.18	0.14	0.12	0.10	0.10	0.10	0.10	0.11	0.12	0.14	0.17	0.19*
Total bits	± 3.42	3.42	3.43	3.48	3.50	3.67	3.78	3.93	4.07	4.23	4.42	4.59
Equivalent °C	± 0.027	0.027	0.027	0.027	0.027	0.029	0.030	0.031	0.032	0.033	0.035	0.036

*Estimated

Table 2. Thermistor and instrument at the same temperature.

Error Source	Temperature (°C) 20	15	10	5	0	-5	-10	-15	-20	-25	-30	-35
Display round-off	± 0.26	0.32	0.40	0.50	0.64	0.81	0.42	0.53	0.69	0.90	0.59	0.78
A/D nonlinearity	± 0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
A/D quantizing	± 0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
A/D non-ideal gain	± 1.60	1.64	1.67	1.74	1.76	1.93	2.37	2.53	2.69	2.85	1.40	1.48
Analog PCB offset	± 0.18	0.14	0.12	0.10	0.10	0.10	0.10	0.11	0.12	0.14	0.17	0.19*
Total bits	± 3.04	3.10	3.19	3.34	3.50	3.84	3.89	4.17	4.50	4.89	3.16	3.45
Equivalent °C	± 0.059	0.049	0.040	0.033	0.027	0.024	0.047	0.039	0.033	0.027	0.027	0.022

* Estimated

instrument's temperature. This condition corresponds to water temperature measurements made in the field. The instrument accuracy was calculated at 5°C intervals over the -35°to 20°C range (Table 1). The second condition was to have both thermistor and the instrument temperature the same. The instrument accuracy was again calculated at 5°C intervals over the same -35° to 20°C range (Table 2). This condition corresponds to air temperature measurements made in the field.

The main source of error was the uncertainty associated with regression equations describing the temperature-dependent variations of the A/D board's full-scale voltage. The uncertainty of these regressions, found using the 90% confidence bands, could be reduced by taking more calibration data for the A/D board within the operating temperature range of the instrument.

The error calculations in Table 3 are for CRREL thermistor serial no. 1805, which has a resistance of 5931.5 Ω and a change of 255 Ω/°C at 0°C. Each error was converted to an equivalent number of bits at the A/D board then all the bits were summed to produce the total system error. The total system error in bits was changed to an equivalent temperature. The total system error describes the error band about the actual temperature.

Table 3 contains the results of a calibration done on 9 and 10 January 1985. The instrument was placed in a cold chamber and connected to a known resistance. The temperature of the chamber was set and the instrument allowed to equilibrate at that temperature. The known resistance was used to simulate thermistor no. 1805. The instrument temperatures were held constant at four temperatures between -18.3° and 24.0°C. For each

Table 3. Calibration on 9 and 10 January 1985 (°C).

Simulated	Error*	Simulated	Error*
	Ambient temperature 24.0°C		
-1.00	0.00	1.00	0.00
-0.50	0.00	0.50	0.00
-0.40	0.00	0.40	-0.01 0.00
-0.30	0.00	0.30	-0.01 0.00
-0.20	0.00	0.20	-0.01 0.00
-0.10	0.00	0.10	0.00 -0.01
0.00	0.00		
	Ambient temperature 3.9°C		
-1.00	-0.03	1.00	-0.01 0.00
-0.50	-0.01	0.50	-0.01 0.00
-0.40	-0.02	0.40	-0.01 -0.02
-0.30	0.00 -0.01	0.30	-0.01
-0.20	0.00 -0.01	0.20	-0.01 0.00
-0.10	-0.02 -0.01	0.10	-0.01
0.00	-0.01		
	Ambient temperature -8.3°C		
-1.00	0.00 0.01	1.00	0.01 0.00
-0.50	0.00	0.50	0.00 0.01
-0.40	0.00	0.40	0.00
-0.30	0.00	0.30	0.00
-0.20	0.00	0.20	0.00
-0.10	0.00	0.10	0.00
0.00	0.00		
	Ambient temperature -18.4°C		
-10.0	-0.02	10.0	0.04
-1.00	0.01	1.00	0.01
-0.50	0.01	0.50	0.01
-0.40	0.01 0.00	0.40	0.01
-0.30	0.01	0.30	0.00 0.01
-0.20	0.01 0.00	0.20	0.01 0.02
-0.10	0.01	0.10	0.01 0.02
0.00	0.01 0.00		

*Two listings means that the LCD was continually shifting between them.

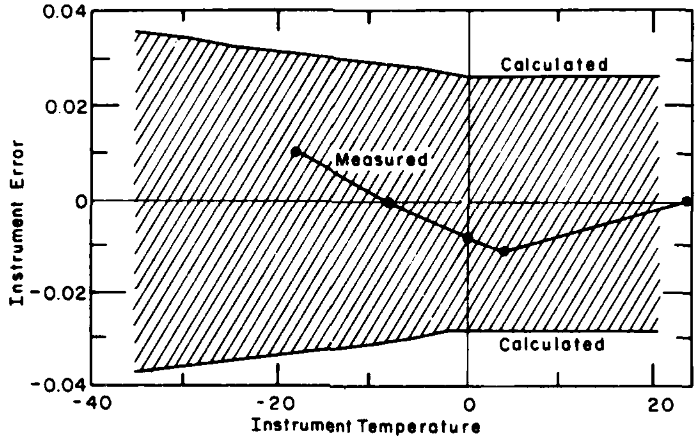


Figure 8. Error analysis (°C).

instrument temperature the simulated thermistor temperatures ranged from -1.0° to 1.0°C.

Figure 8 shows the comparison between the calculated and measured errors with the thermistor held at 0°C while the instrument temperature varied. As can be seen, the measured error was considerably less than the theoretical error.

SUMMARY

The following points would improve the instrument's performance.

1. Use a self-timed EAROM. This would eliminate some modifications to the memory board and the processor board. The time delay circuit could also be eliminated.

2. Add a low battery voltage indication on the display. Two of the available A/D channels could be used to monitor the battery pack voltages directly. A flashing indication would then alert the user to a low battery condition.

3. Reduce the number of battery packs by using a DC/DC converter to develop the needed voltages.

4. Use a 14 or 16-bit A/D for greater accuracy.

5. Add an indicator that shows when the battery pack is indeed charging.

6. Set the gain of the instrumentation op-amp to 10 rather than 1. This can be done easily by grounding the proper instrumentation amplifier pin. This will allow the use of four ranges rather than two on the A/D, improving the accuracy on the upper end of the temperature range.

7. Take more calibration data (especially for the A/D PCB) within the ambient temperature range of interest. This will decrease the uncertainty associated with the regression equations.

LITERATURE CITED

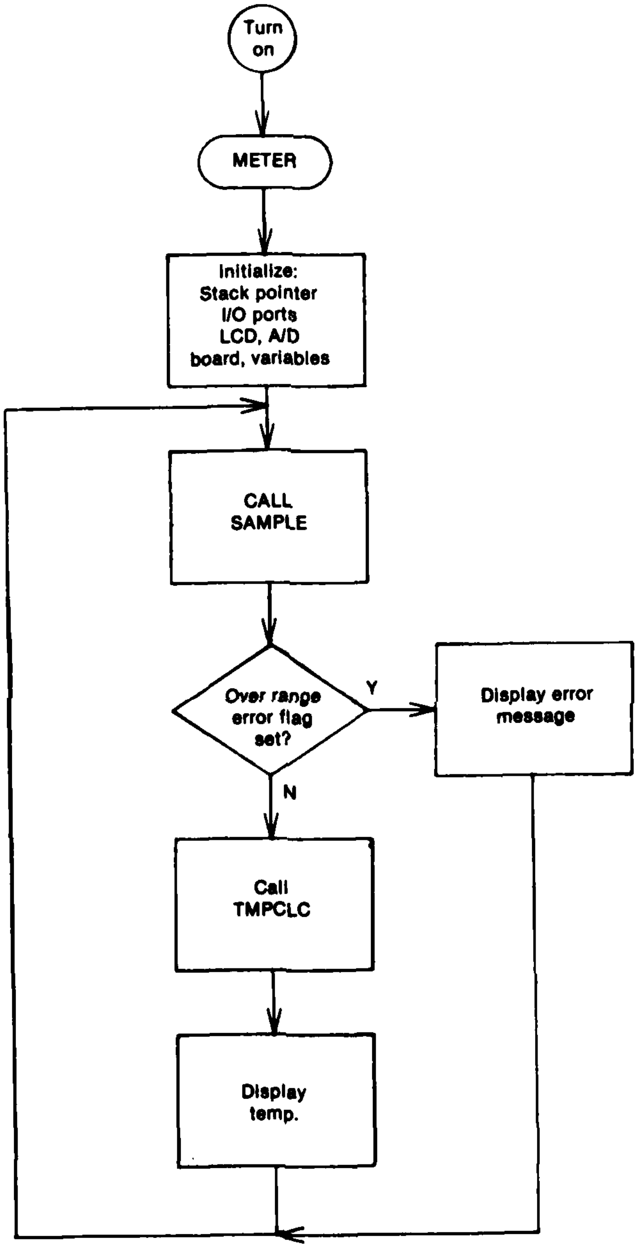
Yellow Springs Instruments Inc. (1971) Thermistor Equation. Yellow Springs, Ohio: YSI, Industrial Division.

Stout, D.F. and M. Kaufman (1976) Handbook of Operational Amplifier Circuit Design. New York: McGraw-Hill.

Steinhart, J.S. and S.R. Hart (1968) Calibration curves for thermistors. Deep Sea Research, 15: 497.

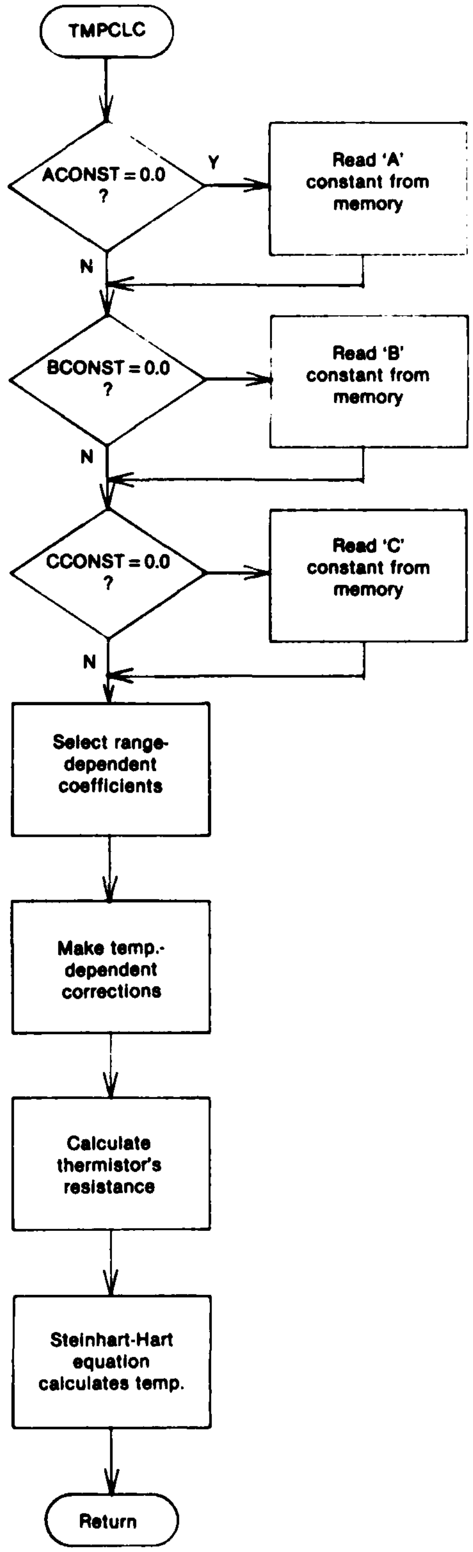
Omega (1984) Temperature Measurement Catalog. Stamford, Connecticut: Omega Engineering, Inc., p. f-9.

APPENDIX A: PROGRAMS



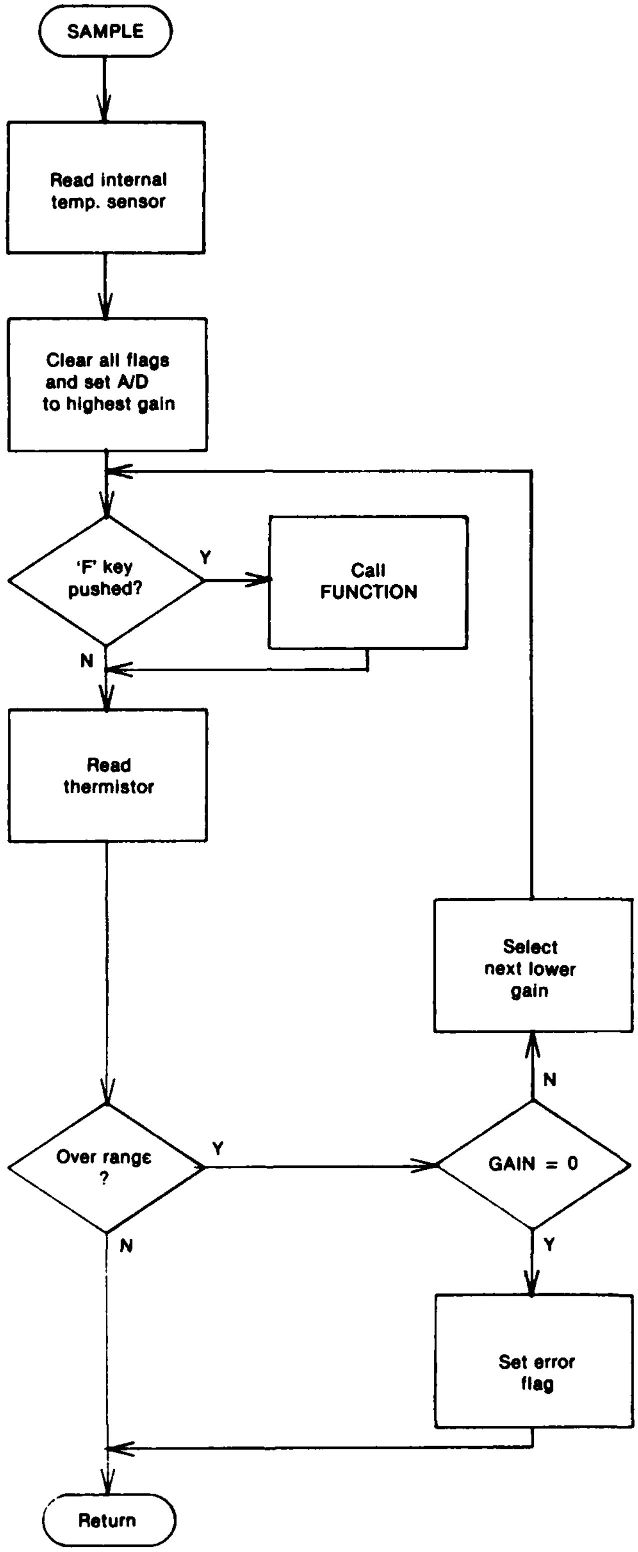
a. METER

Figure A1. Program flow charts.



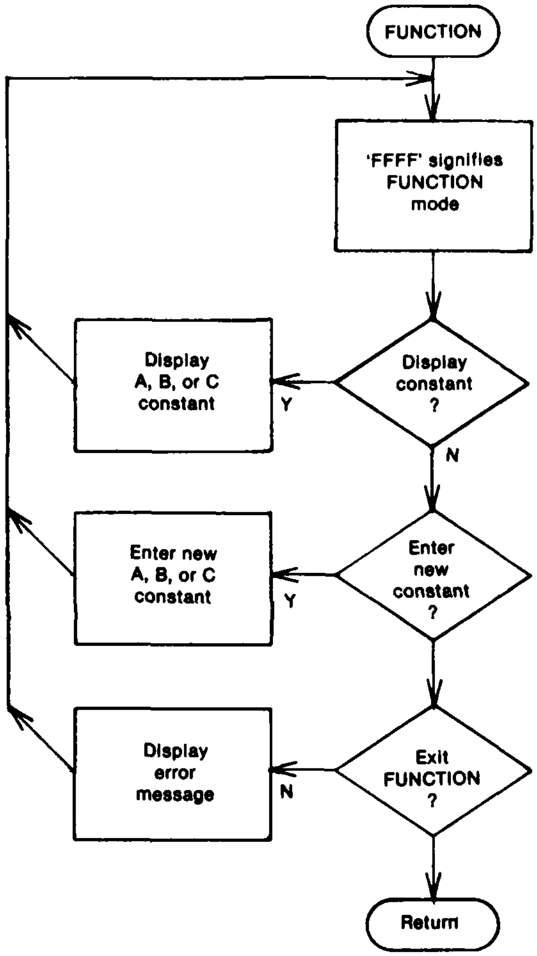
b. TMPCLC

Figure A1 (cont'd). Program flow charts.



c. SAMPLE

Figure A1 (cont'd).



d. FUNCTION.

Figure A1 (cont'd). Program flow charts.

```
/*                                  >>> METER <<<

Written by Gary Trachier USA CRREL
Last Update: 09 January 1985

        This program handles all of the input and output of the uP system
    and interfaces to the arithmetic program.  The hardware requirements
    are the following:

                        Onset Computer MEM-8P 16K memory board
                        Onset Computer CPU-80CA-1 CPU board
                        Quartic Systems CADC-12 12-bit A/D board
                        CRREL prototype Keyboard PC3
                        CRREL prototype LCD PCB

        Some of the above circuit boards have been modified.  Refer to the
    project notebook for specific information.

CODE:   4000-77FF
STACK:  1900-1FFF
HEAP:   1900-1FFF
                                IT APPEARS THAT PASCAL DOES NOT USE ANY SPACE
                                STARTING AT 1900. ONLY 50-100 BYTES ARE USED
                                BY THE STACK AT ANY ONE TIME. */
DATA:   1200-18FF
 */
METER:DO;
        DECLARE SIGN            BYTE;                   /* RETURN SIGN OF READING FROM CHNLCNVRT */
        DECLARE KEYCODE         BYTE;                   /* ACTUAL KEY INPUT 0->F */
        DECLARE LOCATION        ADDRESS;
        DECLARE CONSTANT        BASED LOCATION BYTE;    /* DUMMY TEMPLATE FOR CONSTANTS */
        DECLARE OVERUNDR        BYTE;                   /* OVER/UNDER RANGE INDICATION FROM 'SAMPLE' */
                                                        /* 1000_0000=OVER RANGE */
                                                        /* 0000_0001=UNDER RANGE */
                                                        /* 0000_0000=OK (IN BOUNDS) */

        DECLARE TEMPLO          BYTE AT (1000H);
        DECLARE TEMPHI          BYTE AT (1001H);
        DECLARE TEMPSGN         BYTE AT (1002H);
        DECLARE ZROFLAG         BYTE AT (1005H);        /* 1='ZERO' THE OFFSET TO +0.010, 0=NORMAL OPERATION */
        DECLARE DDRA            BYTE AT (1084H);        /* DDR, PORT A (LCD DATA IN LOWER NIBBLE) */
        DECLARE DDRB            BYTE AT (1085H);        /* DDR, PORT B (LCD DP, SIGN, /CS, DIGIT SELECT) */
        DECLARE DDRC            BYTE AT (1086H);        /* DDR, PORT C (KEYBOARD) */
        DECLARE PORTA           BYTE AT (1080H);        /* ADDRESS OF PORT A */
        DECLARE PORTB           BYTE AT (1081H);        /* ADDRESS OF PORT B */
        DECLARE PORTC           BYTE AT (1082H);        /* ADDRESS OF PORT C */
        DECLARE AFLAG           BYTE AT (7FFDH);        /* EEROM FLAG SHOWING THAT A-CONSTANT IS INITIALIZED */
        DECLARE BFLAG           BYTE AT (7FFEH);        /* EEROM FLAG SHOWING THAT B-CONSTANT IS INITIALIZED */

    DECLARE CFLAG           BYTE AT (7FFFH);        /* EEROM FLAG SHOWING THAT C-CONSTANT IS INITIALIZED */
    DECLARE DISPLAY         ADDRESS;                /* USED TO SEND DATA TO 'LCD' */
    DECLARE ADRDNG          ADDRESS;                /* RAW RESULT OF A/D CONVERSION INCL. FLAGS */
    DECLARE INDEX           ADDRESS;                /* G.P. LOOP COUNTER */
    DECLARE DUMMY           ADDRESS;                /* G.P. VARIABLE */

    DECLARE OFFSETBASE      LITERALLY '7824H';      /* BASE OF THE ZERO OFFSET IN EEROM */
    DECLARE ABASE           LITERALLY '78CCH';      /* BASE OF A-CONSTANT */
    DECLARE BBASE           LITERALLY '79CCH';      /* BASE OF B-CONSTANT */
    DECLARE CBASE           LITERALLY '7819H';      /* BASE OF C-CONSTANT */
    DECLARE SETUPA          LITERALLY '1111_1111B'; /* DIRECTION SETUP, PORT A (LCD DATA) */
    DECLARE SETUPB          LITERALLY '1111_1111B'; /* DIRECTION SETUP, PORT B (LCD CONTROL & SIGNS) */
    DECLARE SETUPC          LITERALLY '0000_0000B'; /* DIRECTION SETUP, PORT C (KEYBOARD) */
    DECLARE ZRO             LITERALLY '00H';
    DECLARE STACKTOP        LITERALLY '1FFFH';      /* TOP OF STACK MEMORY */
    DECLARE FOREVER         LITERALLY 'WHILE 1';
    DECLARE KBD             LITERALLY 'PORTC';      /* LOCATION OF KEYBOARD INTERFACE */
    DECLARE KBDMSK          LITERALLY '0000_1111B'; /* DATA MASK FOR KEYBOARD INPUTS */
    DECLARE KBDVAL          LITERALLY '0001_0000B'; /* DATA VALID BIT MASK FOR KEYBOARD INTERFACE */
    DECLARE ADDATALO        LITERALLY '20H';        /* READ PORT FOR A/D LOW ORDER BYTE */
    DECLARE ADDATAHI        LITERALLY '22H';        /* READ PORT FOR A/D HIGH ORDER BYTE */
    DECLARE ADSTATUS        LITERALLY '24H';        /* READ PORT FOR A/D STATUS */
    DECLARE ADPWR           LITERALLY '20H';        /* WRITE PORT FOR A/D POWER & RESET CONTROL */
    DECLARE ADGAINCHNL      LITERALLY '24H';        /* WRITE PORT FOR A/D CHANNEL & GAIN CONTROL */
    DECLARE POWERON         LITERALLY '0000_0001B'; /* TURN CADC-12 POWER ON */
    DECLARE POWEROFF        LITERALLY '0000_0000B'; /* TURN CADC-12 POWER OFF */
    DECLARE RESETON         LITERALLY '0000_0000B'; /* ACTIVATE RESET ON CADC-12 */
    DECLARE RESETOFF        LITERALLY '0000_0010B'; /* CLEAR RESET ON CADC-12 */

COLDSTART: PROCEDURE INTERRUPT 0;                   /* ROUTINE FOR 'RESET' & RST0 */
           GOTO MAIN;                               /* GOTO START OF MAIN MODULE */
   END COLDSTART;

   TMPCLC: PROCEDURE EXTERNAL;                      /* PASCAL RESISTANCE TO TEMPERATURE ROUTINE */
   END TMPCLC;

   BEGXQQ: PROCEDURE EXTERNAL;                      /* INIT. THE PASCAL INTERNAL POINTERS */
   END BEGXQQ;

       DAA:PROCEDURE (NUM) BYTE;                    /* CONVERT 8-BIT BINARY TO 2-DIGIT BCD */
           DECLARE NUM BYTE ;
           DECLARE LSD BYTE ;
           DECLARE MSD BYTE ;

           LSD=NUM MOD 100;
           MSD=(NUM-LSD)/100;
           RETURN SHL(MSD,4) OR LSD;
   END DAA;

  ZROCONST:PROCEDURE (CMDCODE);                             /* ZEROS ONE OF THE PASCAL-CALCULATED CONSTANTS IN RAM */
           DECLARE CMDCODE BYTE;                            /* INPUT SHOULD HAVE 'A', 'B', OR 'C' IN THE LOWER NIBBLE */
           DECLARE INDEX BYTE;                              /* G.P. LOOP COUNTER */
           DECLARE LOCATION ADDRESS;                        /* ADDRESS TO BE ZEROED */
           DECLARE CONST BASED LOCATION BYTE;               /* ONE-BYTE TEMPLATE FOR CLEARING MEMORY */
```

```
        DECLARE MASK LITERALLY '0000_1111B';        /* LO-NIBBLE MASK */
        DECLARE ACONST LITERALLY '10C6H';           /* BASE FOR 4-BYTE CALCULATED A-CONSTANT */
        DECLARE BCONST LITERALLY '10CAH';           /* BASE FOR 4-BYTE CALCULATED B-CONSTANT */
        DECLARE CCONST LITERALLY '100EH';           /* BASE FOR 4-BYTE CALCULATED C-CONSTANT */

        IF (CMDCODE AND MASK)=0AH THEN
                LOCATION=ACONST;                    /* CLEAR THE A-CONSTANT */
        IF (CMDCODE AND MASK)=0BH THEN
                LOCATION=BCONST;                    /* CLEAR THE B-CONSTANT */
        IF (CMDCODE AND MASK)=0CH THEN
                LOCATION=CCONST;                    /* CLEAR THE C-CONSTANT */
        DO INDEX=1 TO 4;                            /* CLEAR THE 4-BYTE REAL NUMBER */
                CONST=ZRO;                          /* CLEAR THE BYTE */
                LOCATION=LOCATION+1;                /* POINT TO THE NEXT BYTE */
        END;
END ZROCONST;

SETSCLR:PROCEDURE (PORT,BITMSK,VALUE);  /* PORT BIT SET/CLR ROUTINE */
        DECLARE PORT BYTE;              /* 1 FOR PORT A, 2 FOR PORT B, & 3 FOR PORT C */
        DECLARE BITMSK BYTE;            /* BITS TO BE SET OR CLEARED ARE 1, ALL OTHERS ARE 0 */
        DECLARE VALUE BYTE;             /* BITS CORRESPONDING TO THOSE 'SET' IN 'BITS' SHOULD
                                           BE THE DESIRED DATA, IE 1 OR 0, ALL OTHERS ARE IRRELEVANT */
        DECLARE SETDATA BYTE;
        DECLARE CLRDATA BYTE;
        DECLARE PORTASET BYTE AT    (1C8CH);        /* PORT A BIT SET ADDRESS */
        DECLARE PORTACLR BYTE AT    (1C85H);        /* PORT A BIT CLEAR ADDRESS */
        DECLARE PORTBSET BYTE AT    (1C9DH);        /* PORT B BIT SET ADDRESS */
        DECLARE PORTBCLR BYTE AT    (1C89H);        /* PORT B BIT CLEAR ADDRESS */
        DECLARE PORTCSET BYTE AT    (1C95H);        /* PORT C BIT SET ADDRESS */
        DECLARE PORTCCLR BYTE AT    (1C3AH);        /* PORT C BIT CLEAR ADDRESS */

        SETDATA=(BITMSK AND VALUE);                 /* BITS TO BE SET ARE 1 */
        CLRDATA=((BITMSK XOR VALUE) AND BITMSK);/* BITS TO BE CLEARED ARE NOW 1 */
        DO CASE (PORT-1);
                DO;                                 /* PORT A */
                        PORTASET=SETDATA;
                        PORTACLR=CLRDATA;
                END;
                DO;                                 /* PORT B */
                        PORTBSET=SETDATA;
                        PORTBCLR=CLRDATA;
                END;
                DO;                                 /* PORT C */
                        PORTCSET=SETDATA;
                        PORTCCLR=CLRDATA;
                END;
        END;
END SETSCLR;

LCDBYTE:PROCEDURE (DIGIT,ONEDIGIT,DECIMAL,HBAR,VBAR,VALUE);    /* OUTPUTS ONE DIGIT OF DATA & VARIOUS
                                                                  DP, BARS ETC */
        DECLARE DIGIT BYTE;                                    /* DIGIT SELECT (0-3) */
        DECLARE ONEDIGIT BYTE;                                 /* ONEDIGIT ANNUNCIATOR, UNUSED */
        DECLARE DECIMAL BYTE;                                  /* DECIMAL POINT, 1=ON, 0=OFF */
        DECLARE HBAR BYTE;                                     /* HORIZONTAL BAR */



        DECLARE VBAR BYTE;                                     /* VERTICAL BAR */
        DECLARE VALUE BYTE;                                    /* ACTUAL DIGIT'S VALUE */
        DECLARE DUMMY BYTE;                                    /* TEMPORARY VARIABLE */

        DECLARE ONEMSK LITERALLY '0000_0001B';                 /* HALF-DIGIT '1' MASK */
        DECLARE DPMSK LITERALLY '0000_0001B';                  /* DECIMAL POINT MASK */
        DECLARE HBMSK LITERALLY '0000_0001B';                  /* HORIZONTAL BAR MASK */
        DECLARE VBMSK LITERALLY '0000_0001B';                  /* VERTICAL BAR MASK */
        DECLARE VALMSK LITERALLY '0000_1111B';                 /* NUMERIC DATA MASK */

        VALUE=VALUE AND VALMSK;                                /* SAVE ONLY THE LOWER NIBBLE */
        DIGIT=(DIGIT XOR 0000_0011B);                          /* CHANGE TO 7211'S NUMBERING SYSTEM */
        PORTA=VALUE;                                           /* SEND THE HEXADECIMAL NUMBER TO PORTA */
        DUMMY=SHL(DIGIT,1);
        CALL SETSCLR(2,0000_0110B,DUMMY);                      /* SET UP THE DIGIT SELECT BITS */
        CALL SETSCLR(2,0000_0001B,0000_0000B);                 /* THEN SEND AN ACTIVE-LOW PULSE */
        CALL SETSCLR(2,0000_0001B,0000_0001B);                 /* TO DS\ ON THE 7211 */
        DUMMY=SHL((ONEDIGIT AND ONEMSK),7);
        CALL SETSCLR(2,1000_0000B,DUMMY);                      /* UPDATE '1' DIGIT */
        DUMMY=SHL((DECIMAL AND DPMSK),6);
        CALL SETSCLR(2,0100_0000B,DUMMY);                      /* UPDATE DECIMAL POINT */

        DUMMY=SHL((HBAR AND HBMSK),5);
        CALL SETSCLR(2,0010_0000B,DUMMY);                      /* UPDATE HORIZONTAL BAR */

        DUMMY=SHL((VBAR AND VBMSK),4);
        CALL SETSCLR(2,0001_0000B,DUMMY);                      /* UPDATE VERTICAL BAR */
END LCDBYTE;

MSSDELAY: PROCEDURE (MS);                            /* 10-MS DELAY ROUTINE */
          DECLARE COUNT ADDRESS;
          DECLARE COUNT2 ADDRESS;
          DECLARE MS BYTE;

          DECLARE MSSDLY LITERALLY '005CH';          /* LOOP CONSTANT */

          DO COUNT=1 TO MS;
                  DO COUNT2=0 TO MSSDLY;
                          ;                          /* 10-MS BLOCK */
                  END;
          END;
END MSSDELAY;

DELAY:    PROCEDURE (SECONDS);                       /*  1 SECOND DELAY ROUTINE */
          DECLARE SECONDS BYTE;
          DECLARE COUNT ADDRESS;
          DECLARE COUNT2 ADDRESS;

          DECLARE DLY LITERALLY '2FE9H';             /* LOOP CONSTANT */

          DO COUNT=1 TO SECONDS;
                  DO COUNT2=0 TO DLY;
                          ;                          /*  1 SECOND BLOCK */
```

```
                END;
        END;
END DELAY;

CHNLCNVRT:PROCEDURE (CHANNEL,GAIN,MODE);          /* DOES CONVERSION FOR A SINGLE CHANNEL & GAIN */
        DECLARE GAIN    BYTE;                      /* 0->3 */
        DECLARE CHANNEL BYTE;                      /* 0->15 SINGLE-ENDED, 0->7 DIFFERENTIAL */
        DECLARE MODE    BYTE;                      /* 0=DIFFERENTIAL, 1=SINGLE-ENDED */
        DECLARE DUMMY   ADDRESS;

        DECLARE ACTIVE  LITERALLY '0000_0001B';           /* A/D STATUS, 1=CONVERTING 0=INACTIVE */
        DECLARE SIGNMSK LITERALLY '1000_0000B';           /* 1=POSITIVE, 0=NEGATIVE */
        DECLARE DATAMSK2 LITERALLY '00001111_11111100B';  /* DATA MASK WITH THE LOWER BITS AS "DON'T CARE" */

        IF MODE=0 THEN
                MODE=000_01_000B;                  /* DIFFERENTIAL */
        ELSE
                MODE=000_10_000B;                  /* SINGLE-ENDED */
CNVRT2:OUTPUT(ADGAINCHNL)=SHL(GAIN,5D) OR MODE OR CHANNEL;     /* SETUP GAIN, MODE & CHANNEL THEN DO A
                                                                  CONVERSION */
        DO WHILE (INPUT(ADSTATUS) AND ACTIVE)=ZRO;
                ;       /* WAIT FOR A/D STATUS BIT TO GO HIGH... */
        END;
        DO WHILE (INPUT(ADSTATUS) AND ACTIVE)=01H;
                ;       /* ...THEN LOW. */
        END;
        DUMMY=INPUT(ADDATAHI);                          /* GET THE UPPER BITS & FLAGS */
        SIGN=ROL((DUMMY AND SIGNMSK),1);                /* EXTRACT THE SIGN BIT */
        DUMMY=SHL(DUMMY,8D);                            /* SHIFT TO UPPER BYTE */
        DUMMY=DUMMY OR INPUT(ADDATALO);                 /* THEN GET THE LOWER BYTE */
        IF (ADRDNG AND DATAMSK2) <> (DUMMY AND DATAMSK2) THEN
                DO;
                        ADRDNG=DUMMY;           /* SAVE THE LAST READING */
                        GOTO CNVRT2;            /* REPEAT UNTIL THE READINGS CLOSELY AGREE */
                END;
        ADRDNG=DUMMY;                           /* RETURN THE STABILIZED VALUE */
END CHNLCNVRT;

KEYIN:PROCEDURE;                                /* DEBOUNCED SINGLE KEY INPUT ROUTINE */
        DECLARE TMPKEY BYTE;                    /* TEMPORARY RAW KEYCODE STORAGE */
        DECLARE KBDSTBL(16D) BYTE DATA
                (00H,01H,02H,03H,04H,05H,06H,07H,08H,
                09H,0AH,0BH,0CH,0DH,0EH,0FH);          /* CONVERSION TABLE FROM RAW TO REAL KEYCODE */

        DECLARE MASK LITERALLY '0000_1111B';    /* MASK OUT UNUSED BITS */

        DO WHILE (KBD AND KBDVAL)=KBDVAL;       /* WAIT FOR LAST KEY TO BE RELEASED */
                ;
        END;
        CALL MSDELAY (10);                      /* DELAY FOR 10-MS */
KEYIN1:DO WHILE (KBD AND KBDVAL)=ZRO;   /* WAIT FOR NEXT KEY INPUT */
                ;
        END;
        TMPKEY=KBD;                     /* GRAB THE RAW KEYCODE FROM MEMORY-MAPPED KEYBOARD PORT */
        CALL MSDELAY (10);              /* DELAY FOR 10-MS */



        IF TMPKEY<>KBD THEN
                GOTO KEYIN1;                    /* IF KEY BOUNCED THEN READ AGAIN */
        KEYCODE=KBDSTBL(TMPKEY AND MASK);       /* LOOKUP THE ACTUAL KEYCODE */
END KEYIN;                                      /* AND RETURN TO CALLER. */

LCD:    PROCEDURE (ONEDIGIT,DECIMAL,HBAR,VBAR,VALUE);
        DECLARE ONEDIGIT BYTE;
        DECLARE DECIMAL BYTE;
        DECLARE HBAR BYTE;
        DECLARE VBAR BYTE;
        DECLARE VALUE ADDRESS;                  /* CODED AS 4 BCD DIGITS, AAAA_BBBB_CCCC_DDDD */
        DECLARE INDEX BYTE;                     /* G.P. LOOP COUNTER */
        DO INDEX=0 TO 3;                        /* OUTPUT DATA TO EACH LCD DIGIT */
                CALL LCDBYTE (INDEX,ONEDIGIT,DECIMAL,HBAR,VBAR,VALUE);
                VALUE=SHR(VALUE,4);
        END;
END LCD;

RTERROR:PROCEDURE INTERRUPT 1;                  /* THIS SHOULD START AT 0008H */
        DO FOREVER;
                CALL LCD (ZRO,ZRO,ZRO,ZRO,0EEEEH);
                CALL DELAY (10);
                CALL LCD (ZRO,ZRO,ZRO,ZRO,0000H);
                CALL DELAY (10);
        END;
END RTERROR;

FUNCTION:PROCEDURE;                             /* INVOKED BY 'F' KEY FROM KEYBOARD */
        DECLARE CONSTSARRAY ARRAY(16D) BYTE;    /* ARRAY FOR ENTERING THE CONSTANTS */
        DECLARE INDEX BYTE;                     /* LOOP COUNTER */
        DECLARE INDEX2 BYTE;                    /* LOOP COUNTER */
        DECLARE CMDCODE BYTE;                   /* WILL BECOME DA, DB, DC, EA, EB, EC, OR EE */
        DECLARE BASE2 BYTE;                     /* POINTER FOR THE "CONSTANT" ARRAY */
        DECLARE BASE ADDRESS;                   /* POINTS TO BASE OF THE CONSTANT TO BE CHANGED/DISPLAYED */

        DO FOREVER;                             /* DO UNTIL BROKEN OUT THROUGH 'EE' COMMAND */
                KEYCODE=ZRO;
                CMDCODE=ZRO;
                DISPLAY=0FFFFH;                         /* PUT 'FFFF' ON DISPLAY... */
                CALL LCD (ZRO,ZRO,ZRO,ZRO,DISPLAY);     /* TO SIGNIFY 'FUNCTION' MODE. */
                CALL KEYIN;                             /* GET THE FIRST KEY */
                CMDCODE=SHL(KEYCODE,4);                 /* AND MOVE IT TO THE UPPER NIBBLE */
                DISPLAY=KEYCODE;
                CALL LCD (ZRO,ZRO,ZRO,ZRO,DISPLAY);     /* KEY 1 TO LCD */
                CALL KEYIN;                             /* GET THE SECOND KEY */
                CMDCODE=CMDCODE OR KEYCODE;             /* COMBINE UPPER AND LOWER NIBBLES */
                DISPLAY=CMDCODE;
                CALL LCD (ZRO,ZRO,ZRO,ZRO,DISPLAY);     /* KEY 2 TO LCD */
                CALL MSDELAY (500);                     /* 500-MS DELAY */

                DO CASE CMDCODE;        /* NOW EXECUTE THE COMMAND */

                    218:                /* DA */
                    219:                /* DB */
                    220:                /* DC */
```

```
     DO;
             IF KEYCODE=0AH THEN
                     BASE=ABASE;                 /* POINT TO THE A CONSTANT */
             IF KEYCODE=0BH THEN
                     BASE=BBASE;                 /* POINT TO THE B CONSTANT */
             IF KEYCODE=0CH THEN
                     BASE=CBASE;                 /* POINT TO THE C CONSTANT */
             DO  INDEX=0 TO 2;                    /* GROUP COUNTER */
                     DO INDEX2=0 TO 3;            /* DIGIT COUNTER */
                             LOCATION=BASE+(4*INDEX)+(3-INDEX2);       /* CALC. THE OFFSET */
                             CALL LCDBYTE (INDEX2,ZRO,ZRO,ZRO,ZRO,CONSTANT);       /* DISPLAY
                                     A GROUP OF 4
                     END;
                     CALL KEYIN;         /* AND WAIT FOR USER TO PROMPT FOR THE NEXT GROUP OF 4
             END;
     END;

234:                    /* EA */
235:                    /* EB */
236:                    /* EC */
     DO;
             IF KEYCODE=0AH THEN
                     DO;
                             AFLAG=0FFH;         /* CLEAR THE BYTE */
                             AFLAG=0AAH;         /* THEN INIT. IT */
                             BASE=ABASE;         /* ENTER THE A CONSTANT */
                     END;
             IF KEYCODE=0BH THEN
                     DO;
                             BFLAG=0FFH;
                             BFLAG=0BBH;
                             BASE=BBASE;         /* ENTER THE B CONSTANT */
                     END;
             IF KEYCODE=0CH THEN
                     DO;
                             CFLAG=0FFH;
                             CFLAG=0CCH;
                             BASE=CBASE;         /* ENTER THE C CONSTANT */
                     END;
             DO INDEX=0 TO 3;
                     CONST$ARRAY(INDEX)=KEYCODE;      /* FILL FIRST 4 PLACES WITH A,B,C, OR D
             END;
             DO INDEX=4 TO 15;
                     CONST$ARRAY(INDEX)=ZRO;          /* THEN THE REST WITH ZEROS. */
             END;
             DO INDEX=0 TO 3;
                     DISPLAY=SHL(DISPLAY,4) OR CONST$ARRAY(INDEX);
             END;
             BASE2=0;                  /* INIT. THE ARRAY POINTER */
             DO WHILE BASE2 < 11D;     /* DIGIT COUNTER */
                     KEYCODE=CFH;
                     CALL LCD (ZRO,ZRO,ZRO,ZRO,DISPLAY);
                     DO WHILE (KEYCODE>9) AND (KEYCODE<>0BH);         /* WAIT FOR 0-9 OR 'B'
                                                                          KEY INPUT */
                             CALL KEYIN;


                             END;
                             IF KEYCODE<10D THEN              /* DIGITS 0-9 */
                                     DO;
                                             CONST$ARRAY(BASE2+4)=KEYCODE; /* SAVE THE DIGIT */
                                             BASE2=BASE2+1;  /* INC. THE ARRAY POINTER */
                                             DISPLAY=SHL(DISPLAY,4D);
                                             DISPLAY=DISPLAY OR KEYCODE;     /* PUT THE DIGIT ON THE
                                                                                DISPLAY */
                                     END;
                             IF KEYCODE=0BH AND (BASE2>0) THEN        /* BACKUP A DIGIT-AT-A-TIME */
                                     DO;
                                             BASE2=BASE2-1;  /* DEC. THE ARRAY POINTER */
                                             DO INDEX=0 TO 3;        /* BACKUP ONE DIGIT &
                                                                        RESTRUCTURE 'DISPLAY'
                                                     DISPLAY=SHL(DISPLAY,4) OR
                                                             CONST$ARRAY(BASE2+INDEX);
                                             END;
                                     END;
                     END;
                     CALL LCD (ZRO,ZRO,ZRO,ZRO,DISPLAY);       /* SHOW THE LAST DIGIT */
                     CALL DELAY (10);                          /* 1-SECOND DELAY */
                     DO INDEX=0 TO 11;                         /* WRITE THE ARRAY INTO EEROM */
                             LOCATION =BASE+INDEX;             /* CALC. THE DESTINATION ADDRESS */
                             CONSTANT=CFFH;                    /* CLEAR THE EEROM BYTE... */
                             CONSTANT=CONST$ARRAY(INDEX+4);        /* THEN WRITE THE ACTUAL DIGIT */
                     END;
                     CALL ZROCONST (CMDCODE);                  /* CLEAR THE REAL-TYPE TO ALERT PASCAL
                                                                  ROUTINE */
             END;
         238:                /* EE */
                     RETURN;          /* RETURN TO TEMPERATURE MEASURE MODE */
         240:                /* F0 */
             DO;
                     ZROFLAG=1;          /* DO AN OFFSET ZERO (+0.01C) NEXT TIME TMPCLC IS CALLED */
                     DO LOCATION=OFFSETBASE TO OFFSETBASE+3;
                             CONSTANT=CFFH;  /* ERASE THE EEROM SO A NEW OFFSET CAN BE WRITTEN */
                     END;
                     RETURN;             /* GO BACK TO TEMPERATURE MEASURE MODE */
             END;

             OTHERWISE DO;               /* THIS CODE IS FOR ALL INVALID COMMANDS */
                     CALL LCD (ZRO,ZRO,ZRO,ZRO,0EEEEH);       /* SHOW 'EEEE' AS AN ERROR CONDITION */
                     CALL DELAY (10);                         /* WAIT FOR 1 SECOND */
                     CALL LCD (ZRO,ZRO,ZRO,ZRO,0000H);        /* THEN ZERO THE DISPLAY */
             END;
         END;
     END;
END FUNCTION;

 SAMPLE:PROCEDURE;            /* DOES CONVERSION OF THERMISTOR & PCB TEMP. SENSOR */
        DECLARE GAIN     ADDRESS AT (1C14H);          /* 0->5 GAIN RANGE */
        DECLARE RONG     ADDRESS AT (1CC3H);          /* 16-BIT SPACE FOR A/D VALUE */
        DECLARE GOODSMPL BYTE;
```

```
        DECLARE AMBTMP  ADDRESS AT (1C12H);                 /* SEND PCB TEMP. TO TMPCLC ROUTINE */

        DECLARE DATAMSK LITERALLY '00001111_11111111B'; /* KEEPS 12 DATA BITS, THROW OUT OV, SIGN ETC. */
        DECLARE OVRFLW  LITERALLY '00010000_00000000B'; /* A/D OVERFLOW BIT, 1=OVERFLOW 0=OK */
        DECLARE NEGPOS  LITERALLY '10000000_00000000B'; /* A/D SIGN BIT, 1=POS 0=NEG */
        DECLARE OVRFLWFLAG LITERALLY '1000_0000B';      /* OVER THE DYNAMIC RANGE OF THE OP-AMP & A/D SYSTEM */

        CALL CHNLCNVRT (0,2,1);                             /* READ THE PCB TEMP. SENSOR */
        AMBTMP=ADRDNG AND DATAMSK;                          /* MASK OUT THE FLAG BITS */
        GAIN=5;                                             /* START WITH MAXIMUM GAIN */
        GOODSMPL=ZRO;
        OVERSUNDR=ZRO;
        DO WHILE GOODSMPL=ZRO;                              /* LOOP 'TILL A GOOD READING OR THE LOWEST RANGE IS
                                                               REACHED */

                KEYCODE=ZRO;
                IF (KBD AND KBDVAL)=KBDVAL THEN
                        KEYCODE=KBD AND KBDMSK;             /* GRAB THE KEY BEING PUSHED */
                IF KEYCODE=0FH THEN
                        CALL FUNCTION;                      /* GO INTO THE FUNCTION MODE */

                CALL CHNLCNVRT (1,GAIN,0);                  /* READ THE THERMISTOR */
                RDNG=ADRDNG AND DATAMSK;                    /* SAVE THE DATA BITS */
                IF (ADRDNG AND OVRFLW)=ZRO THEN
                        GOODSMPL=01H;               /* GOOD CONVERSION, NO OVERFLOW */
                ELSE
                        DO;
                                IF GAIN=ZRO THEN
                                        DO;
                                                OVERSUNDR=OVRFLWFLAG;   /* FLAG AS AN OVER RANGE READING ON
                                                                           LOWEST GAIN RANG
                                                GOODSMPL=0FFH;          /* SET FLAG SO WE CAN BREAK OUT OF LOOP
                                        END;
                                ELSE
                                        GAIN=GAIN-1;    /* TRY THE NEXT LOWER GAIN SETTING */
                        END;
        END;
    END SAMPLE;

MAIN:DO;                                /* MAIN CODE SECTION */
        STACKPTR=STACKTOP;              /* SET UP THE STACK POINTER */
        CALL @EGXQC;                    /* SET UP THE PASCAL INTERNALS */
        DDRA=SETUPA;                    /* SET UP THE NSC810 I/O PORTS */
        DDRB=SETUPB;
        DDRC=SETUPC;
        PORTA=ZRO;                      /* CLEAR ANY GLITCHES STORED IN LATCH */
        PORTB=1111_0001B;               /* SET UP DIGIT SELECT & SPECIAL CONTROL BITS ON LCD */
        DUMMY=PORTC;                    /* CLEAR THE KEYBOARD LATCH */
        CALL LCD (1,1,1,1,8888H);               /* TURN ALL DISPLAY SEGMENTS ON... */
        CALL DELAY (10);                        /* AND WAIT 1 SECOND. */
        OUTPUT(ADPWR)=POWERON;  /* POWER UP THE CADC-12 */
        CALL MSSDELAY(50);              /* WAIT FOR 50mS... */
        OUTPUT(ADPWR)=POWERON OR RESETOFF;      /* ...THEN CLEAR THE RESET BIT */
        DUMMY=INPUT(ADSTATUS);  /* CLEAR THE CADC-12 INTERRUPT BIT */
        ZROFLAG=ZRO;                    /* CLEAR THE AUTO ZERO FLAG */











          CALL ZROCONST (0AH);      /* CLEAR THE A CALCULATED CONSTANT */
          CALL ZROCONST (0BH);      /* CLEAR THE B CALCULATED CONSTANT */
          CALL ZROCONST (0CH);      /* CLEAR THE C CALCULATED CONSTANT */

          DO WHILE (AFLAG<>0AAH) OR (BFLAG<>0BBH) OR (CFLAG<>0CCH);
                  CALL FUNCTION;  /* FORCE ENTRY OF NEW CONSTANTS IN NEW EEROM */
                  DO LOCATION=OFFSETBASE TO OFFSETBASE+3;
                          CONSTANT=ZRO;   /* ZERO THE OFFSET IF A NEW EEROM */
                  END;
          END;

          DO FOREVER;
                  CALL SAMPLE;                        /* GET A READING */
                  IF OVERSUNDR>ZRO THEN
                          CALL LCD (ZRO,ZRO,ZRO,ZRO,0EEEEH);      /* DISPLAY AN ERROR INDICATION */
                  ELSE
                          DO;
                                  CALL TMPCLC;                        /* CONVERT RESISTANCE TO TEMPERATURE */
                                  DISPLAY=DAA(TEMPHI);                        /* COMBINE THE UPPER... */
                                  DISPLAY=SHL(DISPLAY,8) OR DAA(TEMPLO); /* AND LOWER 2 DIGITS */
                                  CALL LCD (ZRO,1,1,NOT TEMPSGN,DISPLAY); /* AND OUTPUT W/ DP & SGN TO DISPLAY */
                          END;
          END;
      END MAIN;
END METER;
```

```
(*                              >>> TMPCLC  (V 1.0) <<<

Written by Gary Trachier USA CRREL
Last updated:  08 January 1985

          This program interfaces to the main code named 'METER'.
        It is called as an untyped procedure in METER.  This routine
        uses the 12-bit number from the A/D, the thermistor constants
        that are stored in EAROM, and the gain setting of the instrumentation
        op-amp as inputs.  It then uses the Steinhart, Hart equation to
        calculate the temperature.  The temperature is returned to the
        main code in TEMPLO and TEMPHI.  TEMPLO is an 8-bit representation
        of the two digits to the right of the decimal point.  This will
        range from 0 to 99.  TEMPHI is a value from 0 to 99 corresponding
        to the two digits to the left of the decimal point.  The sign
        of the temperature is returned in TEMPSIGN.

          This module must be compiled  by the Starplex Pascal compiler.
        It then must be linked to the skeleton Pascal library and the
        relocatable code from 'METER'.  Refer to the compilation and
        linkage instructions for specifics.

 *)
PROGRAM DUMMY;

VAR ERTLOC [PUBLIC], ERCLOC [PUBLIC] : WORD;
PROCEDURE INIFOC [PUBLIC]; BEGIN END;
PROCEDURE ENDYOC [PUBLIC]; BEGIN END;
PROCEDURE PTYLOQ (LEN: WORD; SRC: ADSMEM) [PUBLIC]; BEGIN END;
FUNCTION  GTYLOQ (LEN: WORD; DST: ADSMEM): WORD [PUBLIC];
   BEGIN GTYLOQ := 0 END;
PROCEDURE PLYLOQ [PUBLIC]; BEGIN END;
PROCEDURE ENDLOC [PUBLIC]; BEGIN END;
PROCEDURE INILOQ [PUBLIC]; BEGIN END;

PROCEDURE TMPCLC [PUBLIC];

     CONST
        AMULT = 1.0E-03;          (* MULTIPLIERS FOR THE THERMISTOR *)
        BMULT = 1.0E-04;          (* CONSTANT MANTISSAS. *)
        CMULT = 1.0E-07;

        ACOEF2 = 0.746597E-03;   (* CONSTANTS FOR THE ANALOG PCB TEMPERATURE *)
        BCOEF2 =-0.945213E-06;   (* CORRECTION.  THESE ARE FOR SERIAL# 001 *)
        CCOEF2 = 0.371049E-09;
        DCOEF2 =-0.406270E-13;

     VAR

        A1CONST    [ORIGIN 16#7800] : SINT;       (* A-CONSTANT, DIGIT 1  *)
        A2CONST    [ORIGIN 16#7801] : SINT;       (* A-CONSTANT, DIGIT 2  *)
        A3CONST    [ORIGIN 16#7802] : SINT;       (* A-CONSTANT, DIGIT 3  *)
        A4CONST    [ORIGIN 16#7803] : SINT;       (* A-CONSTANT, DIGIT 4  *)
        A5CONST    [ORIGIN 16#7804] : SINT;       (* A-CONSTANT, DIGIT 5  *)
        A6CONST    [ORIGIN 16#7805] : SINT;       (* A-CONSTANT, DIGIT 6  *)

A7CONST    [ORIGIN 16#7806] : SINT;       (* A-CONSTANT, DIGIT 7  *)
A8CONST    [ORIGIN 16#7807] : SINT;       (* A-CONSTANT, DIGIT 8  *)
A9CONST    [ORIGIN 16#7808] : SINT;       (* A-CONSTANT, DIGIT 9  *)
A10CONST   [ORIGIN 16#7809] : SINT;       (* A-CONSTANT, DIGIT 10 *)
A11CONST   [ORIGIN 16#780A] : SINT;       (* A-CONSTANT, DIGIT 11 *)

B1CONST    [ORIGIN 16#780C] : SINT;       (* B-CONSTANT, DIGIT 1  *)
B2CONST    [ORIGIN 16#780D] : SINT;       (* B-CONSTANT, DIGIT 2  *)
B3CONST    [ORIGIN 16#780E] : SINT;       (* B-CONSTANT, DIGIT 3  *)
B4CONST    [ORIGIN 16#780F] : SINT;       (* B-CONSTANT, DIGIT 4  *)
B5CONST    [ORIGIN 16#7810] : SINT;       (* B-CONSTANT, DIGIT 5  *)
B6CONST    [ORIGIN 16#7811] : SINT;       (* B-CONSTANT, DIGIT 6  *)
B7CONST    [ORIGIN 16#7812] : SINT;       (* B-CONSTANT, DIGIT 7  *)
B8CONST    [ORIGIN 16#7813] : SINT;       (* B-CONSTANT, DIGIT 8  *)
B9CONST    [ORIGIN 16#7814] : SINT;       (* B-CONSTANT, DIGIT 9  *)
B10CONST   [ORIGIN 16#7815] : SINT;       (* B-CONSTANT, DIGIT 10 *)
B11CONST   [ORIGIN 16#7816] : SINT;       (* B-CONSTANT, DIGIT 11 *)

C1CONST    [ORIGIN 16#7818] : SINT;       (* C-CONSTANT, DIGIT 1  *)
C2CONST    [ORIGIN 16#7819] : SINT;       (* C-CONSTANT, DIGIT 2  *)
C3CONST    [ORIGIN 16#781A] : SINT;       (* C-CONSTANT, DIGIT 3  *)
C4CONST    [ORIGIN 16#781B] : SINT;       (* C-CONSTANT, DIGIT 4  *)
C5CONST    [ORIGIN 16#781C] : SINT;       (* C-CONSTANT, DIGIT 5  *)
C6CONST    [ORIGIN 16#781D] : SINT;       (* C-CONSTANT, DIGIT 6  *)
C7CONST    [ORIGIN 16#781E] : SINT;       (* C-CONSTANT, DIGIT 7  *)
C8CONST    [ORIGIN 16#781F] : SINT;       (* C-CONSTANT, DIGIT 8  *)
C9CONST    [ORIGIN 16#7820] : SINT;       (* C-CONSTANT, DIGIT 9  *)
C10CONST   [ORIGIN 16#7821] : SINT;       (* C-CONSTANT, DIGIT 10 *)
C11CONST   [ORIGIN 16#7822] : SINT;       (* C-CONSTANT, DIGIT 11 *)

OFFSET     [ORIGIN 16#7824] : REAL;       (* ZERO OFFSET FOR THE SYSTEM *)

TEMPLO     [ORIGIN 16#1000] : SINT;       (* LOWER TWO DIGITS OF RESULTING TEMPERATURE *)
TEMPHI     [ORIGIN 16#1001] : SINT;       (* UPPER TWO DIGITS OF RESULTING TEMPERATURE *)
TEMPSGN    [ORIGIN 16#1002] : SINT;       (* SIGN OF TEMPERATURE, 1=NEG 0=POS *)

NUMBER     [ORIGIN 16#1003] : INTEGER;    (* 16-BIT RESISTANCE PASSED FROM THE PLM MAIN-MODULE *)
AMPGAIN    [ORIGIN 16#1014] : INTEGER;    (* GAIN SETTING OF PROGRAMMABLE OP-AMP *)

BUILD,TEMP,X,Y,RES,VFS         : REAL;    (* LOCAL VARIABLES *)
C1,MVTRUE,MVACC,RANGE          : REAL;
OFFSETM,OFFSETB                : REAL;
ACOEF1,BCOEF1,CCOEF1,DCOEF1: REAL;        (* COEFFICIENTS FOR A/D SPAN DRIFT DUE TO TEMP. *)
ZROFLAG    [ORIGIN 16#1005] : SINT;       (* 0=USE STORED OFFSET, 1=CALC. & STORE NEW OFFSET *)
ACONST     [ORIGIN 16#1006] : REAL;       (* RESULTANT A CONSTANT *)
BCONST     [ORIGIN 16#100A] : REAL;       (* RESULTANT B CONSTANT *)
CCONST     [ORIGIN 16#100E] : REAL;       (* RESULTANT C CONSTANT *)
AMBTMP     [ORIGIN 16#1012] : INTEGER;    (* READING FROM THE ON-BOARD TEMPERATURE SENSOR *)

BEGIN
  IF ACONST=0.0 THEN
         BEGIN
                 BUILD := 0.0;
                 BUILD := A1CONST + (A2CONST / 10.0) + (A3CONST / 100.0) + (A4CONST / 1000.0) + (A5CONST /
                          10000.0);
```

```
                    BUILD := BUILD + (A6CONST / 10000.0) + (A7CONST / 100000.0) + (A8CONST / 1000000.0) +
                            (A9CONST / 10000000.0);
                    BUILD := BUILD + (A10CONST / 100000000.0) + (A11CONST / 1000000000.0);
                    ACONST:= BUILD * AMULT;                (* FINAL RESULT *)
            END;

    IF BCONST=0.0 THEN
            BEGIN
                    BUILD := 0.0;
                    BUILD := B1CONST + (B2CONST / 10.0) + (B3CONST / 100.0) + (B4CONST / 1000.0) + (B5CONST /
                            1000.0);
                    BUILD := BUILD + (B6CONST / 10000.0) + (B7CONST / 100000.0) + (B8CONST / 1000000.0) +
                            (B9CONST / 10000000.0);
                    BUILD := BUILD + (B10CONST / 100000000.0) + (B11CONST / 1000000000.0);
                    BCONST:= BUILD * BMULT;                (* FINAL RESULT *)
            END;

    IF CCONST=0.0 THEN
            BEGIN
                    BUILD := 0.0;
                    BUILD := C1CONST + (C2CONST / 10.0) + (C3CONST / 100.0) + (C4CONST / 1000.0) + (C5CONST /
                            1000.0);
                    BUILD := BUILD + (C6CONST / 10000.0) + (C7CONST / 100000.0) + (C8CONST / 1000000.0) +
                            (C9CONST / 10000000.0);
                    BUILD := BUILD + (C10CONST / 100000000.0) + (C11CONST / 1000000000.0);
                    CCONST:= BUILD * CMULT;                (* FINAL RESULT *)
            END;

    CASE AMPGAIN OF
            0:BEGIN                                        (* 5.0 VOLT RANGE *)
                    RANGE :=  5.0;                         (* THIS RANGE IS UNUSED *)
                    OFFSETM:= 7.561E-03;                   (* NUMBER OF BITS DRIFT IN A/D OFFSET BETWEEN -40 & +20 *)
                    OFFSETB:=-13.85;
                    ACOEF1:=  5.25372;
                    BCOEF1:=  0.758045E-04;
                    CCOEF1:= -0.165753E-06;
                    DCOEF1:=  0.407301E-10;
              END;

            1:BEGIN                                        (* 2.5 VOLT RANGE *)
                    RANGE :=  2.5;                         (* THIS RANGE IS UNUSED *)
                    OFFSETM:= 7.561E-03;                   (* NUMBER OF BITS DRIFT IN A/D OFFSET BETWEEN -40 & +20 *)
                    OFFSETB:=-14.01;
                    ACOEF1:=  2.93996;
                    BCOEF1:= -0.375653E-03;
                    CCOEF1:=  0.969300E-07;
                    DCOEF1:= -0.533937E-11;
              END;

            2:BEGIN                                        (* 1.0 VOLT RANGE *)
                    RANGE :=  1.0;
                    OFFSETM:= 7.561E-03;                   (* NUMBER OF BITS DRIFT IN A/D OFFSET BETWEEN -40 & +20 *)
                    OFFSETB:=-14.49;
                    ACOEF1:=  1.12125;
                    BCOEF1:= -0.814063E-04;



                    CCOEF1:=  0.914955E-08;
                    DCOEF1:=  0.204027E-11;
              END;

            3:BEGIN                                        (* 0.5 VOLT RANGE *)
                    RANGE :=  0.5;
                    OFFSETM:= 7.561E-03;                   (* NUMBER OF BITS DRIFT IN A/D OFFSET BETWEEN -40 & +20 *)
                    OFFSETB:=-15.29;
                    ACOEF1:=  0.503223;
                    BCOEF1:=  0.317428E-04;
                    CCOEF1:= -0.269064E-07;
                    DCOEF1:=  0.550908E-11;
              END;

            4:BEGIN                                        (* 0.25 VOLT RANGE *)
                    RANGE :=  0.25;
                    OFFSETM:= 7.561E-03;                   (* NUMBER OF BITS DRIFT IN A/D OFFSET BETWEEN -40 & +20 *)
                    OFFSETB:=-16.89;
                    ACOEF1:=  0.320247;
                    BCOEF1:= -0.103811E-03;
                    CCOEF1:=  0.391343E-07;
                    DCOEF1:= -0.483480E-11;
              END;

            5:BEGIN                                        (* 0.10 VOLT RANGE *)
                    RANGE :=  0.10;
                    OFFSETM:= 7.561E-03;                   (* NUMBER OF BITS DRIFT IN A/D OFFSET BETWEEN -40 & +20 *)
                    OFFSETB:=-21.69;
                    ACOEF1:=  0.114595;
                    BCOEF1:= -0.145211E-04;
                    CCOEF1:=  0.380318E-08;
                    DCOEF1:= -0.217719E-12;
              END;
END;

VFS := ACOEF1 + (BCOEF1 * AMBTMP) + (CCOEF1 * AMBTMP * AMBTMP) + (DCOEF1 * AMBTMP * AMBTMP * AMBTMP); (* ACTUAL
                                                                                                  FULL SCALE VOLTAGE

Y := NUMBER + (OFFSETM * AMBTMP + OFFSETB);      (* CALC. THE OFFSET DRIFT FOR THE A/D & CORRECT 'NUMBER' *)

MVADC := (Y/4095)*VFS;                           (* CALC. THE ACTUAL VOLTAGE GOING INTO THE A/D CONVERTER *)

                                                 (* ANALOG PCB CORRECTION FACTOR FOLLOWS... *)
C1 :=ACOEF2 + (BCOEF2 * AMBTMP) + (CCOEF2 * AMBTMP * AMBTMP) + (DCOEF2 * AMBTMP * AMBTMP * AMBTMP);
MVTRUE := MVADC + C1;                            (* CALC. THE ACTUAL mV ACROSS THE THERMISTOR *)

RES := MVTRUE / 12.207E-06;                      (* FINALLY CONVERT mV TO RESISTANCE *)

Y := LN(RES);
X := ACONST + BCONST * Y + CCONST * Y * SQR(Y);        (* STEINHART, HART EQUATION *)
TEMP := 1.0 / X - 273.15;                        (* CONVERT TO CELSIUS *)

IF ZROFLAG = 1 THEN
        BEGIN
                OFFSET := TEMP - 0.01;  (* CALC. OFFSET FOR THERMISTOR IN TRIPLE-POINT CELL *)
```

```
                    ZROFLAG := 0;            (* RESET THE FLAG *)
                END;

        TEMP := TEMP - OFFSET;               (* REMOVE THE SYSTEM OFFSET *)

        IF TEMP >= 0.0 THEN
                TEMPSGN := 0                 (* POSITIVE TEMPERATURE *)
        ELSE
                TEMPSGN := 1;                (* NEGATIVE TEMPERATURE *)

        TEMP := ABS(TEMP);                   (* CHANGE TO POSITIVE VALUE... *)
        TEMP := TEMP + 0.005;                (* ROUND TO 2 DECIMAL PLACES. *)
        TEMPHI := TRUNC(TEMP);               (* UPPER TWO DIGITS OF TEMPERATURE *)
        TEMP := TEMP - TEMPHI;               (* KEEP THE FRACTIONAL PART *)
        TEMPLO := TRUNC(TEMP * 100.0);       (* LOWER TWO DIGITS OF TEMPERATURE *)

END;                    (* END OF TEMPCALC PROCEDURE *)
BEGIN END.              (* BEGIN & END OF 'MAIN MODULE' *)
```

```
                        >>> RAM.MAC <<<
N8080
PUBLIC  ACORQQ, AC2RQQ, AC3RQQ, AC4RQQ, BEGHQQ
PUBLIC  CLNEQQ, CSXEQQ, CUPHQQ, ECNJQQ, EFGXQQ
PUBLIC  ENDHQQ, FACRQQ, FRPXQQ, GAPHQQ, HDRFQQ
PUBLIC  HDRVQQ, INTHQQ, J1QRQQ, J2QRQQ, J3QRQQ
PUBLIC  J4QRQQ, LCNJQQ, POSRQQ, PS2RQQ, PNUXQQ
PUBLIC  RECEQQ, REFEQQ, REPEQQ, RESEQQ, RETXQQ
PUBLIC  SPSJQQ, STKBQQ, STKHQQ, XX1GQQ, XX2GQQ
PUBLIC  XX3GQQ, XX4GQQ, $MMRY

dseg

FRPXQQ:DS2
ACORQQ:DS1
FACRQQ:DS2
AC2RQQ:DS1
AC3RQQ:DS1
AC4RQQ:DS1
BEGHQQ:DS2
CLNEQQ:DS2
CSXEQQ:DS2
CURHQQ:DS2
ENDHQQ:DS2
J1QRQQ:DS1
J2QRQQ:DS1
J3QRQQ:DS1
J4QRQQ:DS1
POSRQQ:DS2
PS2RQQ:DS2
PNUXQQ:DS2
RECEQQ:DS2
REFEQQ:DS2
REPEQQ:DS2
RESEQQ:DS2
RETXQQ:DS2
STKHQQ:DS2
XX1GQQ:DS2
XX2GQQ: DS      2
XX3GQQ:DS2
XX4GQQ: DS      2

ECNJQQ:DS1

CSEG
LCNJQQ:DB ECNJQQ-FRPXQQ

        DW      0
 $MMRY: DW      1900H   ; fill in with lowest AVAILABLE ram location

DSEG
EFGXQQ:DS1
CSEG
GAPHQQ:DW128

DSEG
HDRFQQ:DS2
HDRVQQ:DS2
INTHQQ:DS1
STKBQQ:DS2
SPSJQQ:DS2
END
```

```
N8060           >>> SXENTX.MAC <<<

        TITLE   Entx8 - entry / exit code for Microsoft 8080 Pascal
        SUBTTL  Starplex-II version
;
; System and procedure entry and exit point routines
;
; William Fox,  April 1982
;
; Globals
;
        PUBLIC  $START
        PUBLIC  BEGXQQ,BRTEQQ,ENDXQQ,ERTEQQ,FXSHQQ
;
        EXTRN   BEGHQQ,BEGOQQ,CSXEQQ,CURHQQ,EFGXQQ
        EXTRN   ENDHQQ,ENDOQQ,ENDUQQ,ENDYQQ,ENTGQQ
        EXTRN   FRPXQQ,GAPHQQ,HDRFQQ,HDRVQQ,INIUQQ
        EXTRN   INTHQQ,PNUXQQ,RECEQQ,REFEQQ,REPEQQ
        EXTRN   RESEQQ,STKBQQ,STKHQQ
;
        EXTRN   $MMRY
;
; Defined symbols
;
MEMTP   SET     1FFFH                   ;Top of available memory (note constants are octal)
EXIT    SET     0006H                   ;adr of exit routine
;
;
$START: JMP     BEGXQQ
;
        SUBTTL  Begxqq - system entry point and initialization routine
;
; Main system entry point and initialization routine
;
;     1) Init the stack pointer, frame pointer, and stkbqq.
;     2) Init EFGXQQ to zero.
;     3) Init the heap unit, ie. set beghqq, curhqq, endhqq, and stkhqq.
;     4) Init the machine error context by zeroing reseqq.
;     5) Init the source error context by zeroing csxeqq.
;     6) Init the unit initialization system by zeroing pnuxqq.
;     7) Init the file system by zeroing hdrfqq and hdrvqq.
;     8) Init the machine dependent file system by calling iniuqq.
;     9) Call the escape initialization routine begoqq.
;    10) Call the main program at entgqq.
;
        DB      C
BEGXQQ: POP     B                       ;(GMT) return address of PLM code
        LXI     H,0+0
        SHLD    CSXEQQ
        SHLD    FRPXQQ                  ;init the frame pointer
        SHLD    PNUXQQ
        SHLD    RESEQQ
        SHLD    HDRFQQ



        SHLD    HDRVQQ
        LXI     H,((MEMTP-2) AND 177776O)+0;HL := highest address -2 on even
                                        ;boundary
        SPHL                            ;init the stack pointer
        PUSH    B                       ;(GMT) put return address of PLM code on the new stack
        DCX     H
        DCX     H
        SHLD    STKBQQ                  ;upon stack-heap collision, restore the sp from this
                                        ;  also used by rfpgqq for long jumps
;
; EFGXQQ is used by ENDXQQ as a 'have I been here before' flag, to avoid
;   recursive errors with ENDOQQ.
;
        XRA     A
        STA     EFGXQQ
;
; Init the heap unit
;

        LHLD    $MMRY
        INX     H                       ;round it to even
        MOV     A,L
        ANI     376O
        MOV     L,A
        MVI     M,0                     ;mark the first block as free
        SHLD    BEGHQQ
        SHLD    CURHQQ
        SHLD    ENDHQQ
        CALL    FXSHQQ                  ;init stkhqq
;
        CALL    INIUQQ                  ;init the machine dependent file system
        RET                             ;(GMT) return to the PLM main program
        CALL    BEGOQQ                  ;allow the user to init things
        CALL    ENTGQQ                  ;call the main program
        JMP     ENDXQQ                  ;all finished
        SUBTTL  Brteqq - entry point helper for $runtime
;
; procedure brteqq (offset: word)
;
; $Runtime entry point helper - if this is the top level entry into the
;   runtime system, store the user routine's error context into for emseqq.
;
        DB      C
;
;
;
;
BRTEQQ: POP     H                       ;HL := my return address
        XTHL                            ;HL := offset to my caller's return address
        XCHG                            ;DE := offset to my caller's return address
        LDA     RESEQQ                  ;have I already done this?
        MOV     B,A
        LDA     RESEQQ+1
        ORA     B
        JNZ     BRTX                    ;yes
```

```
        MOV     A,C                     ;DE := - of offset tc my caller's return address
        CMA
        MOV     C,A
        MOV     A,E
        CMA
        MOV     E,A
        INX     C
        LHLD    FRPXCQ                  ;HL := my caller's frame pointer
        DAD     C                       ;HL := address of my caller's return address
        MOV     E,M                     ;DE := my caller's return address
        INX     H
        MOV     D,M
        INX     H
        XCHG
        SHLD    REPECQ                  ;store the user's pc
        XCHG
        MOV     E,M                     ;DE := my caller's saved frame pointer
        INX     H
        MOV     D,M
        XCHG
        SHLD    REFECQ                  ;save the user's frame pointer
        LXI     H,0+C                   ;the 8080 is an unsegmented machine...
        SHLD    RECSECQ                 ;  sc save 0 as the segment base
        LHLD    FRPXCQ                  ;my caller's caller's stack pointer is equal
        LXI     D,2+C                   ;  to my caller's frame pointer + 2
        DAD     D
        SHLD    RESECQ
SRTX:   RET
                                        ;end of ife compil
        SUBTTL  Endxcq - system exit point and termination routine
;
; Main system termination routine
;
;    1)  If EFGXQC is zero, invoke the escape termination routine ENDOCQ.
;    2)  Close all open files by calling endyqc.
;    3)  Invoke the machine dependent file system termination routine, enduqq.
;    3)  Return to Starclex by a jump to EXIT
;
        DB      C
ENDXCQ: LDA     EFGXCQ
        ORA     A
        JNZ     ENX1
        INR     A
        STA     EFGXCQ
        CALL    ENDOCQ
ENX1:   CALL    ENDYCQ
        CALL    ENDUCQ
        JMP     EXIT+0
        SUBTTL  Ertecq - exit point helper for $runtime
;
; procedure ertecq;
;
; $Runtime exit point helper - if I'm returning from the top level
;   entry into the runtime system, restore the error context
;



        DB      C
;
;
;
;
ERTECQ: LHLD    RESECQ                  ;HL := error context sp
        XCHG                            ;DE := c(resecq)
        LHLD    FRPXCQ                  ;HL := current fp
        INX     H                       ;HL := my caller's caller's sp
        INX     H
        MOV     A,L                     ;if sp greater than or equal to resecq,
        SUB     E                       ;  zero error context
        MOV     A,H
        SBB     D
        JC      ERTX
        LXI     H,0+C
        SHLD    RESECQ
ERTX:   RET
                                        ;end of ife compil
        SUBTTL  Fxshqq - public subroutine to compute stkhqq
;
; STKHQC is used to detect collision (within GAPHCQ) of the stack and
;   the heap;  if the current sp + STKHCC does not produce a carry, they're
;   too close.  STKHCQ should always equal - (ENDHQQ + GAPHQC).
;
FXSHCC: LHLD    ENDHCQ
        XCHG                            ;DE := endhqq, HL := scratch
        LHLD    GAPHCQ
        DAD     D                       ;HL := endhqq + gaphcc
        MOV     A,L                     ;negate it
        CMA
        MOV     L,A
        MOV     A,H
        CMA
        MOV     H,A
        INX     H                       ;make it two's complement
        SHLD    STKHCQ
        MVI     A,377Q                  ;release the heap protection semaphore
        STA     INTHCC
        RET
        END     $START
;
        END

;release the heap prct
```

APPENDIX B: INSTRUCTIONS FOR OPERATION OF THERMISTOR METER VERSION 1.0

Getting started

1. Connect a thermistor to the connector on the top panel. Make sure all connections are clean and dry. If the instrument is submerged while in use, remove the thermistor connector and make sure that the pins are dry. Any water could cause a leakage path around the thermistor and have the effect of raising the measured temperatures slightly. This effect will be more pronounced when low temperatures are being measured.
2. Turn the CHARGE/OFF/ON switch to the ON position.
3. If the constants for this thermistor have been entered, the actual temperatures will be displayed. If different constants are required then follow the Entering constants procedure. If "FFFF" is on the display then follow the Entering constants procedure from step 2. (You will not be allowed to exit the FUNCTION mode until all three constants have been entered.)

Entering constants

1. Enter "F" to invoke the FUNCTION mode. "FFFF" should be on the display.
2. Enter "EA,", "EB" or "EC" to enter a new A, B or C constant respectively. For example to enter a new "B" constant simply type "EB."
3. The display will now show "AAAA," "BBBB" or "CCCC." The 11 digits must now be entered, with trailing zeroes if needed. The display will scroll to the left as they are entered. If you make a mistake you may back up a digit at a time by entering "B". This will cause the display to scroll to the right. Any digits that roll off the right end of the display are lost and must be reentered. After entering the eleventh digit you will be returned automatically to function mode signified by "FFFF" on the display.
4. Repeat steps 2 and 3 as needed.
5. The constants may be verified by following the procedure Displaying constants from step 2.
6. Return to the temperature mode by entering "EE."

Displaying constants

1. ENTER "F" to get into the FUNCTION mode. "FFFF" should be on the display now.

2. Enter "DA," "DB" or "DC" to display the A, B or C constant. The first four digits of the mantissa will be displayed. You may look at the rest of the digits by pressing any key. Each keypress will bring up the next group of four digits until all 12 have been seen. The third keypress will bring you back to the function mode, signified by "FFFF" on the display.

3. Return to the temperature mode by entering "EE."

NOTE: If you make a mistake and enter an illegal command, the display will blink "EEEE" at you briefly then return to "FFFF". You may now try again.

There are rechargeable batteries in the meter. They may be charged in the following manner:

1. Turn the CHARGE/OFF/ON switch to the OFF position.

2. Locate the line cord. It has a three-pin grounded plug on one end and an eight-pin Bendix connector on the other.

3. Mate this Bendix connector to the one on top of the instrument and lock it in place. Make sure the pins are clean and dry.

4. Plug the other end of the line cord into a grounded 110-Vac outlet.

5. Turn the CHARGE/OFF/ON switch to the CHARGE position.

The batteries should be fully charged in about 8 hours. The meter should run continuously for 6-8 hours under normal conditions. The battery life may be less when the ambient temperature is below -20°C.

CAUTION: The meter has been sealed completely to waterproof it. Removal of the top panel or turning of any screw heads on the outside of the box will break this seal.

☆ U. S. GOVERNMENT PRINTING OFFICE: 1986--600-048--22,078

END

DTIC

5-86